POLITECNICO DI MILANO

Computer Science and Engineering

Implementation and Test Deliverable

Customers Line-up

Software Engineering 2 Project

Academic year 2020 - 2021

07 February 2021

Version 1.0

Authors:
Samuele NEGRINI
Giorgio PIAZZA

Professor:
Matteo Giovanni ROSSI

Revision history

Date	Revision	Notes
07/02/2021	v.1.0	First release.

Contents

Chapter 1

Introduction

1.1 Purpose

This document is the Implementation and Testing Document for the Customers Line-Up system. Here will be listed the implemented requirements, the used frameworks and the testing done

1.2 Scope

Its aim is to define and report about the software implementation and test procedures defined for all the released software. Precisely, it verifies that the CLup Platform as a whole satisfies its functional requirements. To this end, we define a set of tests covering the different uses that have been defined in the RASD and DD. The main goal of these tests is to specify for each tested demonstrator scenario a series of actions leading to an expected result.

1.3 Glossary

1.3.1 Definitions

Term	Definition
Customers	Identifies the store customers.
Employees	Used in this document to mean both entrance-staff and cashiers.
Store Pass	General term that comprehends both tickets and bookings.
Ticket	Pass generated from the system which is comprehensive of the Queue number and QR code.
Booking, Reservation	Pass generated from the system as a result of the reservation process.
Queue number	Identify user's position in the queue.
QR code	Type of matrix barcode, used by the system for the ticket validation.
System	Totality of the hardware/software applications that contribute to provide the service concerned. Also referred as CLup, Application, Platform.

1.3.2 Acronyms

Acronyms	Term
CLup	Customers Line-up
RASD	Requirements Analysis and Specification Document
DD	Design Document
QR	Quick Response
GPS	Global Positioning System
UI	User Interface
API	Application Programming Interface
OS	Operating System
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
DB	DataBase
DBMS	DataBase Management System

1.3.3 Abbreviations

Abbreviations	Term
e.g.	Exempli gratia
i.e.	Id est
R	Requirement

1.4 Reference documents

- Project assignment specification document.
- CLup, Requirements Analysis and Specification Document.
- CLup, Design Document.
- Course slides on beep.

1.5 Document structure

This document is presented as it follows:

1. **Introduction**: the purpose of this document.
2. **Implemented Requirements**: the requirements and functions that are actually implemented in the software.

3. **Adopted Development Frameworks**: includes adopted development frameworks with references to sections in the DD. It presents adopted frameworks and programming languages with their advantages and disadvantages.
4. **Structure of the Source Code**: explains how the source code is structured both in the front end and in the back end.
5. **Testing**: provides the main test cases applied to the the application.
6. **Installation instructions**: explains how to install and deploy the application.
7. **Effort Spent**: keeps track of the time spent to complete this document. The first table defines the hours spent as a team for taking the most important decisions, the seconds contain the individual hours

Chapter 2

Implemented Requirements

Below are listed the requirements implemented in the prototype. They have been chosen since they are essential for engaging a future user of the system.
Those marked with a star (*) have not been implemented because they are not requested for a group of two people.

2.0.1 Customer

R.1 The system shall allow customers to line-up remotely in a store queue.

Implemented.

R.2 The system shall generate a new ticket when a customer enters a queue.

Implemented.

R.3 The system shall allow customers which do not have a smartphone to get a ticket in place.

Not implemented. Although it is an important feature, it has not been implemented as we do not have the necessary hardware for printing the tickets.

R.4 The system shall allow customers to view the number of people lined up in a queue.

Implemented.

R.5 The system shall give customers an estimated waiting time.

Implemented. We provide a basic functionality since a complex estimation of time would have required too much effort.

R.6 The system shall fetch the GPS position while the user has retrieved a store pass.

Not implemented. We do not consider this one as a vital functionality for a prototype of the app.

R.7 The system shall allow customers to leave a queue.

Implemented.

R.8 The system shall allow customers to filter stores by name.

Implemented.

R.9 The system shall notify customers when it's time to leave for the store.

Not implemented. We do not consider this one as a vital functionality for a prototype of the app.

R.10 The system shall allow customers to book-a-visit to the store and send them the confirmation link and receipt via email.

Not implemented.*

R.11 The system shall allow book-a-visit customers to specify the main categories of item they intend to buy.

Not implemented.*

R.12 The system shall allow customers to delete a store pass.

Partially implemented (only tickets can be deleted).*

R.13 The system shall notify customers when a ticket or booked visit is deleted.

Not implemented.*

R.14 The system shall accept bookings based onto the already booked category items.

Not implemented.*

2.0.2 Store manager

R.15 The system shall allow a registered store manager to login by using their credentials.

Implemented.

R.16 The system shall allow store managers to view the current status of people inside the store.

Implemented.

R.17 The system shall allow store managers to view the current status of people in the queue.

Implemented.

R.18 The system shall allow store managers to view the booked visits to the store.

Not implemented.*

R.19 The system shall allow store managers to set a maximum cap of people inside the store.

Implemented.

R.20 The system shall allow store managers to delete tickets and booked visits.

Partially implemented (only tickets can be deleted).*

2.0.3 Store employee

R.21 The system shall allow a registered store employee to login by using their credentials.

Implemented.

R.22 The system shall allow store employee to view the current status of people inside the store.

Implemented.

R.23 The system shall allow store employee to view the current status of people in the queue.

Implemented.

R.24 The system shall allow store employee to scan QR codes.

Implemented.

R.25 The system shall allow store employee to validate store passes.

Implemented.

2.0.4 CLup admin

R.26 The system shall allow CLup admins to register new supermarkets.

Implemented.

R.27 The system shall generate new manager and staff credentials for each supermarket registered.

Implemented.

Chapter 3

Adopted Development Frameworks

In addition to what has already been said in chapter 5 of the DD, additional libraries and frameworks have been adopted such as:

- **EclipseLink**: provides an extensible framework that allows Java developers to interact with various data services, including databases, web services, Object XML mapping, and Enterprise Information Systems.
- **Maven**: is a build automation tool used primarily for Java projects.
- **Jackson**: is a high-performance JSON processor for Java used to build up messages between the applications and server.
- **Spring Security BCryptPasswordEncoder**: a standard when it comes to password hashing.
- **QR library zxing**: a Java library which allows to display QR codes inside web pages.
- **Bootstrap**: along with his template SBAdmin 2, it has been adopted for the front-end of the web application.
- **jQuery**: JavaScript library to manage AJAX requests and a Bootstrap dependency.

3.1 Programming languages

3.1.1 Java

The Java Programming Language is a general-purpose, concurrent, strongly typed, class-based object-oriented language. It has been used for the server-side development along with Java EE: a set of specifications for enterprise features such as distributed computing and web services. Below are listed some of its advantages and disadvantages:

Advantages

- it is object oriented: in Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- it is platform independent: Java is compiled into platform independent byte code. This byte code is interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- it is robust and mature: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Disadvantages

- Java has high memory and processing requirements. Therefore, hardware cost increases;
- it does not provide support for low-level programming constructs like pointers;
- you don't have any control over garbage collection as Java does not offer functions like delete(), free().

3.1.2 Dart

Dart is a client-optimized language for fast apps on any platform. It has been used as language of the client-side system because of its support of the Flutter framework. Below are listed some of its advantages and disadvantages:

Advantages

- it is object oriented;
- it has a fast VM: programs written in Dart tend to run faster than programs created, for example, in JavaScript;
- it has robust core libraries, baked into the SDK;
- it has a built-in package manager;
- it has useful features like mixins, implicit interfaces, lexical closures, named constructors, one-line functions.

Disadvantages

- it is not a mature language as Java;
- its package collection is not big as competition.

Chapter 4

Structure of the Source Code

4.1 Server side

The server side is divided into two main modules: **CLupEJB** and **CLupWeb**.

4.1.1 CLupEJB

This module encapsulates the business logic of the application. An EJB web container provides a runtime environment for web related software components. Inside, it is divided into:

- **Entities**: these are POJOs representing data that can be persisted to the database. Each entity represents a table stored in the database;
- **Enums**: custom enum types for specifying predefined constants;
- **Exceptions**: custom exception types for specifying errors;
- **Messages**: custom classes for sending data in a known format to the client;
- **Services**: Enterprise Java Bean, the core components of this module which also interacts with the EntityManager to persist data on the database;
- **Util**: other utility classes such as the TokenManager to handle the token authentication without credentials for the customers app.

4.1.2 CLupWeb

This module contains Java servlets that respond to web HTTP requests and qualify as a server-side servlet web API. Inside it is divided into:

- **Controllers**: contains all the web servlets which serve different pages and performs calls on the injected EJB module;
- **Filters**: contains all the web filters used to check user authentication and permissions.

Moreover it includes all the static content like CSS, JS and HTML web pages which will be processed by the template engine Thymeleaf and served to the client web browser.

4.2 Client side

Both the client applications are made with flutter. In flutter, all the logic resides in the *lib* folder. In the *Android* and *iOS* folders there are the respective platform files.

Below is presented the code structure of each app.

4.2.1 Customer App

In the lib folder there are:

- **Enum**: contains the enums.
- **Model**: contains the model classes (such as Ticket, Store etc.).
- **Util**: contains the utility classes (such as the API Manager).
- **Views**: contains all the pages of the app.

4.2.2 Store App

In the lib folder there are:

- **Util**: contains the utility classes (such as the API Manager).
- **Views**: contains all the pages of the app.

Chapter 5

Testing

The testing process has been divided into two type of testing: unit testing and integration testing. Below is shown a summary of the main tests performed.
Due to time constraints, the testing phase was focused on the server side.
No unit tests have been performed on the apps as they are thin clients and most of the logic lives inside the server.

5.1 Unit Testing

5.1.1 OpeningHourService

Add Opening Hour

- Valid insertion of a new list of opening hour from an authorized manager.
- Attempt to insert a new list of opening hour which has timeslots overlapping.
- Attempt to insert a new list of opening hour in which, at least one opening hour, has the timeslot starting time after the end time.
- Attempt to insert a new list of opening hour which has the timeslot end time in the forbidden time (from 23:45 included to 00:00 excluded).
- Attempt to insert a new list of opening hour which has the timeslot start and end time in the forbidden time (from 23:45 included to 00:00 excluded).
- Attempt to insert a new list of opening hours where the timeslots are malformed.
- Attempt to insert a new list of opening hours performed by an authorized manager of a different store.
- Attempt to insert a new list of opening hours performed by an unauthorized user.

Delete Opening Hour

- Valid deletion of list of opening hours performed by an authorized manager.
- Attempt to delete a valid list of opening hours performed by a null user.
- Attempt to delete an invalid list of opening hours performed by an authorized manager.

- Attempt to delete a valid list of opening hours performed by an authorized manager of a different store.
- Attempt to delete a valid list of opening hours performed by an unauthorized user.

Update Opening Hour

- Valid update of the opening hours of a store performed by an authorized manager.
- Attempt to update the opening hours of a store performed by a null user.
- Attempt to update the opening hours of a null store.
- Attempt to update the opening hours of a store with malformed opening hours.

Is In Opening Hour

- Valid check when a time belongs to a today opening hour of a store.
- Invalid check when a time does not belongs to a today opening hour of a store.
- Attempt to check response when a time belongs to an opening hour of a null store.

5.1.2 StoreService

Add Store

- Valid insertion of a new store from a CLup admin.
- Attempt to insert a new store with a non unique name from a CLup admin.
- Attempt to insert a new store with a non unique pec address from a CLup admin.
- Attempt to insert a new store with invalid opening hours from a CLup admin.
- Attempt to insert a new store from an unauthorized user.

Update Store Cap

- Valid update of the store cap of a store from a store manager.
- Attempt to update the store cap of an invalid store from a store manager.
- Attempt to update the store cap of a store from an invalid user.
- Attempt to update the store cap of a store from an unauthorized user.
- Attempt to update the store cap of a store from a manager of another store.

Get Estimate Wait Time

- Valid fetch of the store wait time while the store is not full.
- Valid fetch of the store wait time while the last ticket of the queue is called.
- Valid fetch of the store wait time while the ticket queue is not empty.
- Attempt to fetch the wait time of a not existing store.

5.1.3 TicketService

Add Ticket

- Valid insertion of a new ticket from a customer.
- Attempt to insert a new ticket from a customer to a not existing store.
- Attempt to insert a new ticket from a customer to a closed store.
- Attempt to insert a new ticket from a customer while he already has a ticket.
- Attempt to insert a new ticket from a customer to a store with store cap unset or set to zero.

Delete Ticket

- Valid deletion of a ticket performed by the customer who own the ticket.
- Attempt to delete an invalid ticket performed by the customer who own the ticket.
- Attempt to delete a valid ticket performed by a customer who doesn't own the ticket.
- Valid deletion of a ticket performed by an authorized manager.
- Attempt to delete a valid ticket performed by a null user.
- Attempt to delete an invalid ticket performed by an authorized manager.
- Attempt to delete a valid ticket performed by a non manager user.
- Attempt to delete a valid ticket performed by a manager of another store.

Update Ticket Status

- Valid status update of a VALID ticket performed by an authorized employee.
- Valid status update of an USED ticket performed by an authorized employee.
- Valid status update of default pass code performed by an authorized employee.
- Attempt to update the status of an EXPIRED ticket performed by an authorized employee.

- Attempt to update the status of a not existing ticket performed by an authorized employee.
- Attempt to update the status of a ticket performed by an unauthorized employee.
- Attempt to update the status of default pass code of a not existing store.

5.1.4 UserService

Check Credentials

- Valid check when a user sign in to the platform with a valid usercode and a correct password.
- Invalid check when a user sign in to the platform with a valid usercode and an incorrect password.
- Invalid check when a user sign in to the platform with an invalid usercode.
- Invalid check when a user sign in to the platform with a valid usercode and correct password and more than one user is found in the database.

Generate Credentials

- Valid generation of new manager and employee credentials for a store performed by a CLup Admin.
- Attempt to generate new manager and employee credentials for a store performed by an unauthorized user.
- Attempt to generate new manager and employee credentials for a store performed by a null user.
- Attempt to generate new manager and employee credentials for an invalid store performed by a CLup Admin.

Regenerate Credentials

- Valid regeneration of new manager and employee passwords for a store performed by a CLup Admin.
- Attempt to regenerate new manager and employee passwords for a store performed by an unauthorized user.
- Attempt to regenerate new manager and employee passwords for a store performed by a null user.
- Attempt to regenerate new manager and employee passwords for an invalid store performed by a CLup Admin.

5.2 Integration Testing

After the unit testing phase, we continued with the integration testing phase. In particular, we tested the interfacing of the various modules with each other and the connection with the database. In order not to be too verbose, we will not list each method tested. Tests can be found inside the specific folder inside the project.

5.3 System Testing

A part of the testing phase was also dedicated to the manual testing of web servlets and web pages through the normal usage of a web browser and with the help of a tool for making HTTP POST requests with custom parameters (Postman). In particular, the following tests have been performed successfully:

Admin Dashboard

- Login to website with admin credentials;
- Add a new store and its user credentials;
- Regenerate the credentials for an existing store.
- Display the store list and store details;
- Pass bad input parameters to servlet to check their response.

Manager Dashboard

- Login to website with manager credentials;
- Show the store details;
- Edit the store cap value;
- Edit the store opening hours;
- Display the store passes of the store;
- Delete a store pass;
- Display the default pass code in QR format;
- Pass bad input parameters to servlet to check their response.

Employee Dashboard

- Login to website with employee credentials;
- Display the customer inside the store;
- Display the customer in queue for the store;
- Display the valid store passes to be validated for entry.

Moreover we tested the functionality of the application with the whole system and performed stress tests to check the reliability of the system:

Store App

- Login to the application with employee credentials;
- Scan a valid store pass;
- Scan a used store pass;
- Scan an expired store pass;
- Display the outcome of the store pass scanned.

Customer App

- Display a list of the stores which registered to the system;
- Search for a store by name;
- Display a store details;
- Enter a queue for an open store by generating a new store pass;
- Display the details of a store pass;
- Display the list of all the store passes of a customer.

The system tests were performed also on a iOS device.
Since the process to create a signed a iOS app executable is long and tedious, we only provide the executable of Android apps.

Chapter 6

Installation instructions

6.1 Server

In this section is illustrated the procedure in order to deploy the server on a *Windows* system. The same steps can be applied, with the right adjustments, to deploy on other operating systems.

6.1.1 MySQL

Download *MySQL Community Server* from the official website.
Install it, along with the *MySQL workbench*, following the wizard instructions and set the **root** user credentials. Be sure to store the root password in a safe place.

By default, MySQL is launched as a service at startup.
To check that MySQL is running correctly, try to connect to the server via *MySQL workbench* using the credentials generated.
Once connected to the database server, load the two *SQL Dump* provided in the project folder by doing `File > Run SQL Script...`
You should have two new schemas: `np_clup` and `np_clup_test`.

6.1.2 Java JDK

Now that the database is setup, download the latest *Java JDK* from the official website.
Install it following the wizard instructions and set the environment variables:

- `JAVA_HOME`: set `<path_JDK_installation>`
- `JDK_HOME`: set `<path_JDK_installation>`
- `CLASSPATH`: set `<path_JDK_installation>\lib`
- `PATH`: add `<path_JDK_installation>\bin`

6.1.3 TomEE

Download the latest version of *TomEE plume 8.x.x* from the official website.
Unzip it in a handy location and open the configuration file `tomee.xml` located in the `conf` folder (if it does not exist, please create it).
In this file add the following lines, adjusted with your database credentials:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomee>
    <Resource id="CLupDB" type="DataSource">
        JdbcDriver com.mysql.cj.jdbc.Driver
        JdbcUrl jdbc:mysql://localhost:3306/np_clup
        UserName xxx
        Password xxx
    </Resource>
</tomee>
```

Edit also the `tomcat-users.xml` file located in the `conf` folder and add somewhere in the tag *tomcat-users* the **role** and **user** lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users ...>
    ...
    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <user username="admin" password="password" roles="manager-gui, manager-script"/>
    ...
</tomcat-users>
```

Now set the environment variable `CATALINA_HOME` to the path of *TomEE* folder.

Finally download the Platform Independent Java MySQL Connector from the MySQL website, unzip it and copy the `.jar` in the `lib` folder of *TomEE*.

6.1.4 Maven

Download *Maven* from the official website.
Unzip it in a handy location and open the configuration file `settings.xml` located in the `conf` folder. Add somewhere in the tag *servers*, subtag of *settings*, the **server** lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings ...>
    ...
    <servers>
        ...
        <server>
            <id>TomcatServer</id>
            <username>admin</username>
            <password>password</password>
        </server>
        ...
    </servers>
    ...
</settings>
```

Now add to the environment variable `PATH` the path of *maven* folder.

6.1.5 Server configuration

Before deploying you have two configure two things in the server project.

Store images folder

To set the path of the store images edit the file `web.xml` in the `CLupWeb\src\main\webapp\WEB-INF` folder. Replace the *param-value* of the parameter *upload.location* with the desired folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
    <context-param>
        <param-name>upload.location</param-name>
        <param-value>/clup/uploads</param-value>
    </context-param>
...
</web-app>
```

The example value of `/clup/uploads` corresponds to `C:\clup\uploads`. Be sure to put the provided example images in the specified location.

Testing database credentials

In order to execute the integration testing, there is the need to declare the database user credentials for the test.
Edit the file `persistence.xml` in the `CLupEJB\src\main\resources\META-INF` folder with your database credentials.
You can use the root user or any that has all the privileges on the schema `np_clup_test`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence ...>
    ...
    <persistence-unit name="CLupEJB-testing" transaction-type="RESOURCE_LOCAL">
        ...
        <properties>
            ...
            <property name="javax.persistence.jdbc.user" value="dev"/>
            <property name="javax.persistence.jdbc.password" value="password"/>
            ...
        </properties>
    </persistence-unit>
</persistence>
```

6.1.6 Deploy

In order to deploy the project to TomEE, start the server executing `startup.bat` in the `bin` folder of *TomEE*.
After that, open a terminal in `Server` folder of the project and run `mvn clean install` and then `mvn tomcat7:deploy`.
During this procedures, *Maven* will also perform the `test` goal.
The server should now be deployed. To rerun the server, only the server startup is needed.
You can find the deployed server to http://localhost:8080/clup.

If you want to run the tests you can use `mvn test`.

6.2 Client

6.2.1 APK Installation

To install the apps, download the APK from the repo and install them on your device.
To achieve that, you will be required to allow app installs from unknown sources.

6.2.2 Applications configuration

The first time that you open the app, you will be required to set the server address.
Just write the address where the CLup server is running (e.g. `http://192.168.1.2:8080/clup/`).

Chapter 7

Effort Spent

7.1 Teamwork

Task	Hours
Initial briefing	4
Setup development tools	2
Documentation	4
System testing	10
Document writing	3
Setup building tools	2

7.2 Samuele Negrini

Task	Hours
Server development	45
Setup database	5
Setup testing environment	10
Unit testing	5
Integration testing	15
Document writing	6

7.3 Giorgio Piazza

Task	Hours
Store App development	25
Customer App development	40
Server API development	10
Unit testing	15
Document writing	7