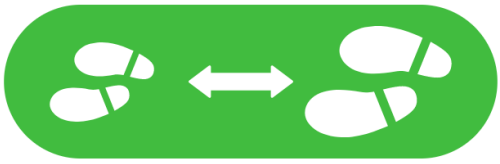
POLITECNICO
MILANO 1863

Computer Science and Engineering

**Software Engineering 2 Project**

# Implementation and Testing Document

Version 1.0 - 07/02/2021



# Customers Line-up

STAY SAFE


Authors

**Enrico Gherardi**
**Ludovico Righi**

Professor

**Elisabetta Di Nitto**

Tutor

**Annalisa Binato**


https://github.com/LudovicoRighi/RighiGherardi

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Scope

This document is a follow up on the previous two, "Requirement Analysis and Specification Document" and "Design Document": it will describe the prototype for Customer LineUp system that has been developed in accordance to them. The document is structured in the following way:

- Chapter 2 : It presents the goals and requirements implemented in the prototype.
- Chapter 3 : It presents the frameworks adopted for the development and testing process.
- Chapter 4 : It presents and explains the structure of the source code.
- Chapter 5 : It gives the information on the testing process.
- Chapter 6 : It gives instructions on how to install and run all the application's components.

## 1.2 Definitions, Acronyms, Abbreviations

This section gives some acronyms in 1.2.1, definitions in 1.2.2 and abbreviation in 1.2.3 which will be used in the document, in order to explain some concept and help the general understanding.

### 1.2.1 Definitions

- Customer: the user of the application who can line-up from his home or can book a visit.
- No-Tech Customer: the Customer of the supermarket that, for external reasons, cannot have an access to CLup in any way and has to enter the store with a ticket.
- Store Manager: the user of the application that can monitor the affluence of his store, have to report to the system all the arriving No-Tech Customers and have to give them the tickets to enter.
- Waiting Time: the time the Customer have to wait before entering the supermarket.
- Real-Time Queue: the system takes into account, managing it as a queue, the customers who had lined-up, who had booked and those without the required technology.
- Long-term Customer: the customer who entered the supermarket using CLup at least 10 times.
- Inferred Duration: the weighted average over the days and hours of the duration of a Long-term Customer's previous visits.
- Line-up Request: the request of a Customer for lining-up from his home in the selected supermarket.
- Booking Request: the request of a Customer to book a Visit in a specific supermarket.
- Visit: the result of a Booking Request.
- Ticket: a number generated by the System that permits to No-Tech Customers to enter the store.

### 1.2.2 Acronyms

- $RASD$: Requirement Analysis and Specification Document.
- $DD$: Design Document.
- $API$: Application Programming Interface.
- $GPS$: Global Positioning System.
- $DBMS$: DataBase Management System.
- $HTTPS$: HyperText Transfer Protocol Secure.
- $TCP/IP$: Transmission Control Protocol / Internet Protocol.
- $RMI$: Remote Method Invocation.
- $UI$: User interface
- $UT$: Unit Testing

### 1.2.3 Abbreviations

- $CLup$: Costumers Line-up.
- $G_N$: Goal number N.
- $R_N$: Requirement number N.

## 1.3 Revision History

- Version 1.0 : 07/02/2021

## 1.4 Reference Documents

- Specification document: "DD Assignment A.Y. 2020/2021".
- Lecture slides of professor M.Rossi and E.di Nitto of Politecnico di Milano.
- Customer Line-up RASD document.

# Chapter 2

# Implemented goals and requirements

In this chapter all the requirements and main functions that are implemented in the prototype are presented. They are listed below:
**Goals:**

- $G_1$ : Customers can line-up from their home to access the supermarket.
- $G_3$ : Customers are notified when they have to go towards the store according the time they need to get to the supermarket.
- $G_4$ : Customers entrances are monitored and managed by the system.

**Requirements:**

- $R_1$ : The System retrieves the GPS position of the Customer.
- $R_2$ : The System sends the QR code for the entrance.
- $R_3$ : The Customer indicates with which mean of transport he intends to go to the supermarket.
- $R_4$ : The Customer indicates the supermarket in which he wants to go.
- $R_5$ : The System shows the Customer his Waiting Time.
- $R_{10}$ : The System computes the time needed for the Customer to reach the supermarket based on his currently GPS position and on his mean of transport.

- $R_{11}$ : The System notifies the Customer when he have to start.
- $R_{12}$ : Customers scan their QR code when they both enter or exit from the supermarket.
- $R_{13}$ : The System computes the Waiting Time considering the Real-Time Queue.
- $R_{17}$ : The System updates the Real-Time Queue.

# Chapter 3

# Development Frameworks

With regard to the Component View presented in the DD, various frameworks and programming languages have been adopted for development, depending on the component and the platform on which it runs on.

## 3.1 Frameworks and Programming Languages

### 3.1.1 Application Server

The language used to develop the Application Server is Java. It has been chosen because it offers an easy scalability and maintainability of the system. In particular, the library JAX-RS has been used to build the APIs of the Restful Server and JPA has been used to manage the connection with the database. Finally, JUnit has been used for the testing.

### 3.1.2 Database

Customer LineUp database is build upon MySQL. It has been chosen for its reliability and its excellent integration with JPA.

### 3.1.3 Mobile Application (For Customers)

Customer LineUp Mobile Application was developed with JavaScript language and with React-Native framework. This choice has been taken because React Native lets create truly native applications for all the main mobile OSs. This choice was made because the catchment area envisaged for CLup covers the entire demographic range and, in this way, we can provide as soon as possible a multi-platform application useful

to people in an emergency period. Furthermore, thanks to its atomic component structure, the development of unit tests is easier and faster.

## 3.2 Middleware Software

**Apache TomEE**: It is a lightweight, yet powerful, JavaEE Application server with feature rich tooling. It permits to build the Restful Server and the connection with the db.
**Expo**: It is a framework and a platform for universal React applications. It is a set of tools and services built around React Native and native platforms that help the programmers develop, build, deploy, and quickly iterate on iOS, Android, and web apps from the same JavaScript codebase.

## 3.3 External API

The Mobile Application uses the Google Map's APIs, in particular:
**GOOGLE MAPS GEOLOCATION API**: to find the exact locations of the Customers.
**GOOGLE MAPS DISTANCE MATRIX API**: to get informations about the recommended route and the duration of the trip (driving, public transportation, walking or cycling) while considering the current traffic situation.

# Chapter 4

# Source Code Structure

This chapter presents how the code has been organized to provide a global picture of the developed classes and components. This is fundamental for facilitating the software maintainance and possible for future extentions such as new functionalities required by the client. Notice that the source code is available at: https://github.com/LudovicoRighi/RighiGherardi/tree/main/IMPLEMENTATION/CLup. The source code of the server is divided into two main folders:

- **CLupEJB**: it comprises the following classes:
  - CLup\CLupEJB\ejbModule\it\clup\entities\Customer.java
  - CLup\CLupEJB\ejbModule\it\clup\entities\LineUpRequest.java
  - CLup\CLupEJB\ejbModule\it\clup\entities\Supermarket.java
  - CLup\CLupEJB\ejbModule\it\clup\entities\StoreManager.java
  - CLup\CLupEJB\ejbModule\it\clup\test\CustomerTest.java
  - CLup\CLupEJB\ejbModule\it\clup\test\LineUpRequestTest.java
  - CLup\CLupEJB\ejbModule\it\clup\test\SupermarketTest.java
- **CLupWEB**: it comprises the following files divided into these two packages:
  - CLup\CLupWEB\build\classes\it\clup\services
    - CLup\CLupWEB\build\classes\it\clup\services\CustomerService.class
    - CLup\CLupWEB\build\classes\it\clup\services\StoreManagerService.class
  - CLup\CLupWEB\build\classes\it\clup\utilities

        * CLup\CLupWEB\build\classes\it\clup\utilities\CustomersLineUp.class
        * CLup\CLupWEB\build\classes\it\clup\utilities\LineUpReqBody.class
        * CLup\CLupWEB\build\classes\it\clup\utilities\Result.class

The source code of the MobileApplication is divided into:

- **ClupExpo**: it comprises only the directories and files:

    - App.js
    - app.json
    - assets
    - babel.config.js
    - components:
        * Header.js
    - models:
        * LineupRequest.js
    - node_modules
    - package-lock.json
    - package.json
    - pages:
        * CreateLineup.js
        * Lineup.js
        * Login.js
        * ShowQRcode.js
        * Signup.js
    - Styles.js
    - Utilities.js

# Chapter 5

# Testing

This chapter present the approach with which testing has been carried out, starting with the planning and the approach that has been adopted in section 5.1 and providing also, in section 5.2 and 5.3, the Unit testing and the Integration test cases.

## 5.1 Testing Plan

As already stated in the Design Document, the approach that has been choice for performing the testing is Bottom-Up: the reasons why this choice has been taken have been described in the DD, but certainly the most important one is the simplicity of the procedure, which typically allows to reach better results in a more efficient way. Moreover, since also the Integration strategy follows a Bottom-Up approach, it is possible to perform testing and integration in parallel with the implementation in a quite straightforward way: in fact, after having implemented a component, Unit testing is perfomed directly by programmers and as the system grows through the Integration testing it is checked that the different modules interacts correclty. At the end of this incremental process, once the whole system is complete, and accurate System testing will ensure that every requirement has been satisfied.

## 5.2 Unit Testing

The Unit testing has been carried out by means of the Junit framework, which is useful for automatizing them; in this way, if they have to be repeated in the future after the introduction of new extentions, the procedure of testing will be very time and effort effective. The test cases are implemented directly in the source code in the

following classes, which can be found at CLup\CLupEJB\ejbModule\it\clup\test: CustomerTest.java, LineUpRequest.java, Supermarket.java, TestRunner.java.
Notice that, in order to reproduce them, it is necessary to have Junit installed.

## 5.3 Integration Testing

In this section are indicated the main Test Cases that have been used in order to perform the Integration testing together with their outcomes. This testing has been performed manually since, for the moment, the developed software is very simple and can be checked immediately. It is however evident that when new functionalities will be introduced, such as the possibility for Customer of doing Bookings and the optimization of the supermarkets based on the departments' affluence, an automatized testing technique will be needed.

### 5.3.1 Test Case 1: Customer Registration

The correct URL string that we have to pass is formatted like:
http:localhost:8080/CLupWEB/signup?username=myUsername&email=myEmail&password=myPassword

For brevity in the table the three parameters are refered to as U, E and P.

| **Test ID** | TC1 |
|---|---|
| **Components** | CustomerService, Customer |
| **Input** | **Output** |
| Missing one or more param between U, E and P | Failed: username, password, email are all mandatory |
| U=(usernameAlreadyInDB) | Failed: the username must be unique |
| E=(emailAlreadyInDB) | Failed: the email must be unique |
| U="Marco", E="marco@gmail.com", P="gatto" | Success |

### 5.3.2 Test Case 2: Customer Login

The correct URL string that we have to pass is formatted like:
http:localhost:8080/CLupWEB/login?username=myUsername&password=myPassword
For brevity in the table the two parameters are refered to as U and P.

| **Test ID** | TC2 |
|---|---|
| **Components** | CustomerService, Customer |
| **Input** | **Output** |
| Missing one or more param between U and P | Failed: username and password are both mandatory |
| U=(invalidUsername) | Failed: the username does not exist |
| U=(validUsername), P=(wrongPassword) | Failed: the password for this username is wrong |
| U=(validUsername), P=(correctPassword) | Success |

### 5.3.3 Test Case 3: Add LineUpRequest

The correct URL string that we have to pass is formatted like:

http:localhost:8080/CLupWEB/addlineupreq?username=myUsername&marketId=myMarketId

For brevity in the table the two parameters are refered to as U and M.

| **Test ID** | TC3 |
|---|---|
| **Components** | CustomerService, Customer, Supermarket |
| **Input** | **Output** |
| Missing one or more param between U and M | Failed: username and marketId are both mandatory |
| U=(invalidUsername) | Failed: the username does not exist |
| M=(invalidMarketId) | Failed: the marketId does not exist |
| U=(userHasAlreadyLinedUp), M=(validMarket) | Failed: the username cannot have more than one LineUpRequest at a time |
| U=(username), M=(validMarket) | Success |

### 5.3.4 Test Case 4: Delete LineUpRequest

The correct URL string that we have to pass is formatted like:

http:localhost:8080/CLupWEB/dellineupreq?requestId=myRequestId

For brevity in the table the parameter is refered to as R.

| **Test ID** | TC4 |
|---|---|
| **Components** | CustomerService |
| **Input** | **Output** |
| Missing param R | Failed: requestId is mandatory |
| R=(invalidRequestId) | Failed: the requestId does not exist |
| R=(validRequestId) | Success |

### 5.3.5 Test Case 5: Customer Entrance

The correct URL string that we have to pass is formatted like:
http:localhost:8080/CLupWEB/entrlineup?requestId=myRequestId
For brevity in the table the parameter is refered to as R.

| **Test ID** | TC5 |
|---|---|
| **Components** | CustomerService |
| **Input** | **Output** |
| Missing param R | Failed: requestId is mandatory |
| R=(invalidRequestId) | Failed: the requestId does not exist |
| R=(validRequestId) | Success |

### 5.3.6 Test Case 6: Retrieve Supermarkets

The correct URL string that we have to pass is formatted like:
http:localhost:8080/CLupWEB/retrmarkets There are no parameters to be passed.

| **Test ID** | TC1.0 |
|---|---|
| **Components** | CustomerService |
| **Input** | **Output** |
| (correct URL string) | Success |

# Chapter 6

# Installation Instructions

## 6.1 Install the Restful Server and the DB

- Install MySQL Workbench 8.0 and MySQL Server.
- Take the file db clup.sql from the folder DatabaseAndConfiguration and execute it with MySQL Workbench to reproduce the database locally.
- Install Eclipse IDE.
- Import the CLuEJB and the CLupWEB as Eclipse project. Make sure to have the file tomee.xml in your Tomee configurations files with the correct username and password of MySQL. Also check to import into the project the eclipseLink.
- Install Tomee Plume 8.0.3
- If you have any problems during these procedures you can find useful resolution tips in the guide, written by Professor Franternali, that you can find in the folder: DatabaseAndConfiguration.
- Start the server, simply by executing the CLupWEB project from Eclipse.
- Moreover, if you do not want to install everything by yourself, we also offer the possibility to test the prototype directly on our server at: "https://08a799d24d68.ngrok.io".

## 6.2 Install the Mobile Application

For completeness, all the CLup tests are done in Android so we suggest you to test the application in this Environment.

### 6.2.1 Run the APK (Recommended)

- In the delivery folder it is provided the APK of the application.
- In the login page, just for the Acceptance test, there is a text input where you have to insert the address of your local server in your LAN and its port. For example, for us is: "http://192.168.1.10:8080".
- Otherwise, you can test the applicatioin with our personal server at the address: "https://08a799d24d68.ngrok.io" as already stated.

### 6.2.2 Run with Expo

- Install expo on your pc with the instruction given from the documentation: "https://docs.expo.io/distribution/building-standalone-apps/"
- Install the Application "Expo Go" on your smartphone.
- Enter with your terminal in the folder CLupExpo.
- Firstly, run the command "npm install –global expo-cli".
- Then, run the command "npm start".
- At this point you will see a web browser page to launch the application. Scan the QR code in your smartphone to download the app and try it.