POLITECNICO
MILANO 1863

CLup - Customer Line Up
Implementation and Testing Document

Project Details
Course: Software Engineering 2

Academic Year: 2020/2021

Reference Professor: Elisabetta Di Nitto

Contributors:
-Ludovica Lerma
-Federico Mainetti Gambera
https://github.com/LudoLe/LermaMainettiGambera.git

TABLE OF CONTENT

1.Introduction

1.A Purpose

Due to the global pandemic going on, some aspects of our everyday life has to be reinvented: avoiding crowding up either inside and outside of grocery shops is one of those.

Clup proposes itself to solve this issue, allowing people who need to go grocery shopping to join the virtual queue of a chosen shop.

Dualy, Clup helps shop owners to monitor the influx of people in their shop helping them to adhere to the current law.

1.B Scope of the document

This document aims to describe the implementation of a prototype of the CLup system, composed of a server and an app client. We will discuss the features of the system that have been implemented, particularly in regards to the features that have been specified in the DD and in the RASD. We will also detail our development choices, including our choice of programming language and frameworks.

The document also contains information about the structure of the repository containing the source code, and the installation instructions.

Finally, we explain the tests that we performed.

2. Implemented features

2.A Reference table

In the following pages,we show a reference table of the ***functional requirements*** for each ***goal*** of the CLup application.

The green color highlights the requirements met by the application.

The red color highlights the requirements which have not been implemented.

The orange color highlights either the requirements which have partially been implemented or the ones which have been implemented with minor changes.

for further details about goals and functional requirements, please read the RASD document.

[G1] Allow manager to sign on the system

ID	REQUIREMENT	DETAILS
R1	A shop owner must be able to begin the sign in process	main functional feature - the interface is UX friendly and easy to use
R2	The system must require the shop owner to provide all the credentials needed	main functional feature - a clear form is provided to the user for they to fill it
R3	the system must check that the credentials don't belong to another account already registered to the system	main security and functional feature
R4	the system must verify the email provided by the shop owner sending a unique code to the address and requesting it from the registration interface	-security feature -not necessary for a prototype version -the email is requested on the registration interface but it is not verified
R5	the system must verify the phone number provided by the shop owner by sending a unique code via SMS and requesting it from the registration interface	-security feature -not necessary for a prototype version -the phone number is requested on the registration interface but it is not verified
R6	The system must verify the credentials through PEC API	-not necessary for a prototype version -the phone number is

		requested on the registration interface but it is not verified
R7	When the sign on process is completed a new account is produced	-main functional feature

[G2] Allow a manager to sign in the system

ID	REQUIREMENT	DETAILS
R8	The manager must be able to begin the sign in process.	main functional feature - the interface is UX friendly and easy to use
R9	The system must require the manager to insert email address and password to authenticate.	main functional feature - a clear form is provided to the user for they to fill it
R10	The system must be check if the credentials inserted match an existing account.	main security and functional feature
R11	The system must present to the manager a solution to reset forgotten credentials.	-security feature and functional feature -not necessary for a prototype version
R12	The system may allow only managers who provide the correct pairs of emails and password to sign in.	-main security feature

[G3] Allow a manager to register their store/stores on the system

ID	REQUIREMENT	DETAILS
R13	The manager must be able to begin process of registering a store	main functional feature - the interface is UX friendly and easy to use

R14	The system must require the manager to provide the address of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the information about their shop
R15	The system must require the manager to provide the schedule of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the schedule of their shop
R16	The system must require the manager to provide the opening days of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the schedule of their shop
R17	The system must require the manager to provide the name of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the information about their shop
R17	The system must require the manager to provide the maximum number of people allowed in the shop	main functional feature - a clear dedicated form is provided to the user in order to register the information about their shop
R18	The manager can optionally provide to the system the subdivision of the shop in areas along with maximum number of people allowed in each area	-not necessary for the prototype to work and not requested
R19	The manager can optionally provide to the system the items sold in their shop	-not necessary for the prototype to work and not requested

R20	The system must communicate the result of the process to the manager	main usability feature

[G5] Allow a manager to check the general status and the statistic of their shops

ID	REQUIREMENT	DETAILS
R27	The manager must be able to begin process of checking the status of an owned shop	main functional feature - the interface is UX friendly and easy to use
R28	The system must be able to retrieve the informations about the number of people enqueued	side functional feature
R29	The system must be able to retrieve the informations about the estimated total duration of the queue	side functional feature
R30	The system must require the manager to provide the opening days of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the schedule of their shop
R17	The system must require the manager to provide the name of the shop	main functional feature - a clear dedicated form is provided to the user in order to register the information about their shop

R17	The system must require the manager to provide the maximum number of people allowed in the shop	main functional feature - a clear dedicated form is provided to the user in order to register the information about their shop
R18	The manager can optionally provide to the system the subdivision of the shop in areas along with maximum number of people allowed in each area	-not necessary for the prototype to work and not requested
R19	The manager can optionally provide to the system the items sold in their shop	-not necessary for the prototype to work and not requested
R20	The system must communicate the result of the process to the manager	main usability feature

[G7] Allow a user to sign on the system

ID	REQUIREMENT	DETAILS
R41	The user must be able to begin the sign on process	main functional feature - the interface is UX friendly and easy to use
R42	The system must require the user to provide all the credentials needed	main functional feature - a clear, simple form is provided to the user for them to fill it
R43	the system must check that the credentials don't belong to another account already registered to the system	main security and functional feature
R44	the system must verify the email provided by the shop owner sending a unique code to the address and requesting it from the registration interface	-security feature -not necessary for a prototype version -the email is requested on the registration interface but it is not verified
R45	the system must verify the phone number provided by the shop owner by sending a unique code via SMS and requesting it from the registration interface	-security feature -not necessary for a prototype version

		-the phone number is requested on the registration interface but it is not verified
R46	When the sign on process is completed and the information provided have been verified by the system, a new account must have been produced	-main functional feature

[G8] Allow a user to sign in the system

ID	REQUIREMENT	DETAILS
R47	The user must be able to begin the sign in process	main functional feature - the interface is UX friendly and easy to use
R48	The system must check if the credentials inserted match an existing account	main functional feature
R49	the system must check that the credentials don't belong to another account already registered to the system	main security and functional feature
R50	the system must present to the user a solution to reset forgotten credentials	-security feature -not necessary for a prototype version
R51	the system may allow only users who provide the corrects pairs of email and password to log in	-core security feature

[G9] Allow a user to search a shop

ID	REQUIREMENT	DETAILS
R52	The user must be able to begin the process of searching a shop	main functional feature - the interface is UX friendly and easy to use
R53	If the user provides his GPS position to the system, the system must be able to ask the user a radius and to retrieve all the shop within that area	-side functionality for better UX
R55	the system must be able to ask the user a keyword to search a shop	main functionality feature for better UX
R56	the system must be able to retrieve all the shops matching the inserted keyword	main functionality feature

R57	the system must be able to present to the user the shops found in a map or in a list	core functionality feature
R58	the system must be able to retrieve general informations of the selected shop and to display them to the user	core functionality feature

[G10] Allow a user to join the virtual queue

ID	REQUIREMENT	DETAILS
R59	After searching a shop, the user must be able to begin the process of joining the virtual queue of a shop	main functional feature - the interface is UX friendly and easy to use - an enqueue button is provided
R60	The system must be able to understand if an enqueuement is possible or if his demand must be rejected, and the system must display this information to the user.	main functionality feature
R61	The system must be able to estimate the queue duration and display this information to the user.	main functionality feature for better UX
R62	The system must require the user to provide the approximate duration of the shopping session and must check whether it's valid.	main functionality feature
R63	If the user provides his GPS position to the system, the system must be able to ask the user by what means it is going to go to the shop and to retrieve the estimated time using Google Maps API. Also the system must be able to send to the user a notification 5 minutes before the estimated time to get to the shop.	side functionality feature for better UX
R64	The system must be able to notificate the user with the waiting time.	core funcionality feature

[G15] Allow a user to retrieve informations about shops

ID	REQUIREMENT	DETAILS
R86	After searching a shop, the user must be able to begin the process of retrieving more informations	functional feature

	about a shop.	
R87	The system must be able to retrieve all the informations about a shop and display them to the user.	functional feature

[G17] Allow a user to exit a previously joined queue

ID	REQUIREMENT	DETAILS
R91	The user must be able to begin the process of exiting a previously joined queue, only if the user is currently enqueued.	main functional feature
R92	The system must be able to correctly remove the user from the queue.	main functional feature
R93	The system must be able to correctly confirm the result of the operation to the user.	main functional feature

[G17] Allow a user to exit a previously joined queue

ID	REQUIREMENT	DETAILS
R91	The user must be able to begin the process of exiting a previously joined queue, only if the user is currently enqueued.	main functional feature
R92	The system must be able to correctly remove the user from the queue.	main functional feature
R93	The system must be able to correctly confirm the result of the operation to the user.	main functional feature

The following list of goals, along with the correlated requirements, have not been implemented, and, thus, their reference table has been omitted.

Some of them have not been implemented because not requested by the specifics (darker red), others because of minor importance referring to the functionalities required by a prototype to work (lighter red).

[G4] Allow a manager to update their shops informations and settings
[G6] Allow a manager to cancel a previously booked shopping session for a customer
[G12] Allow a user to book a shopping session at a grocery store
[G13] Allow a user to retrieve informations about his previously booked visits
[G16] Allow a users and customers to enter and exit stores with QR-codes
[G18] Allow a customer to cancel a previously booked visit from the spot
[G18] Allow a customer to cancel a previously booked visit

[G11] Allow a customer to join a virtual queue from the spot
A special explanation is needed for the choice of not implementing goal [G11]. Indeed, even if this goal concerns the enqueuement process, the way it would be implemented strictly relies on the *book a visit* feature of the application.

3. Development frameworks

3.A Server

The server side consists in the back-end part of Clup and it is based on the JEE framework. We developed three services: AccountManagerService, QueueService, ShopService.

Each service is deployed separately in order to adhere to the microservices structure we declared.

Each service is divided in three main layers: The Web Layer, The Business Layer and The Data Layer.

This kind of framework is very intuitive and facilitates decoupling. It also facilitates conceptual organization of the application. However, since it was our first time developing an application with such structure, it was initially hard to cope with.

The **back-end** part of CLup has been utterly developed in **Java**, since it is the best known developing language by the members of our team.

We chose Intellij as supporting IDE and deployed the application on **TomEE JavaEE Application server.**

3.B Web app

For the client-side of the prototype we decided to create a web application and not a mobile application, in order to accelerate development, as the web environment, as opposed to that of native applications, is better known within the team.

Our past knowledge was limited to the basics of JavaScript, HTML and CSS, but, wanting to raise the quality of the application, we decided to learn and use React.js.

Although this library is new matter for the team, we have chosen it for its fame and to have a solid environment to rely on and to take advantage of all those best practices that emerge naturally from a correct use of this complex JavaScript library .

As expected React supported us very well during production, but it took us a long time to learn and get used to the new concepts and workflow.

The choice of React is also based on the fact that the entire code base is easily translatable into React-Native, so that in the future, native applications can be created for both Android and IOS.

The entire frontend UI was build for mobile screens only, desktop size screens may look bad.

In the Design Document we talked about using Redux to manage the state of the application, but, as the prototype has few features and is relatively simple, using Redux would have been excessive and would probably only have complicated the application structure.

For what concerns the aesthetics of the web app, in order not to waste too much time, we have created only the initial page in a final version, reproducing the mock-ups in the Design Document as precisely as possible.
Having therefore proved that the mock-up designs are achievable, we have opted for a very simple and minimal design for all the other pages. The purpose of this choice is not to waste time in optimizing graphics, but rather to concentrate on data logic and communication with servers.

We used Visual Studio Code as IDE.

4. Structure of the source code

The GitHub repository of the project is

https://github.com/LudoLe/LermaMainettiGambera.git

The folder /code contains all the code we have developed for the Implementation assignment.

4.A Backend

/code/backend is the directory for the Java source code of the server. This folder contains one directory for each of the backend parts described in section 2.B of the DD.

We have a couple packages for each module, one for the web part, one for the business logic:

- /AMW: account manager web
- /AMB: account manager business
- /SSW: shop service web
- /SSB: shop service business
- /QSW: queue service web
- /QSB: queue service business

In each of these directories, the source code is in */src/main/java*, and the code for the tests is in */src/test/java*.

code/backend alsp contains /*DL* and /*UTILS*.:

- */DL* represents the Data Layer and contains the ORM foot print of our database according to JPA specifics, and the low-level services that offer CRUD operations on the entities.
- /UTILS contains the prototypes classes for the requests received by the server and the responses processed by AMW, QSW and SSW, and a specific class used to wrap those answers in a HTTP Response with json body. Those objects have been put in UTILS in order not to duplicate the cose in all of those modules.

4.B Frontend

/code/frontend/clup is the directory for the React source code of the web app.

It follows the regular structure of a React app:

- package.json and package-lock.json: information about all the dependencies of the project.
- /public: contains basic files like the index.html, manifest.json, etc.
- /src: contains all the CLUP web app code:
    - App.js: it is the root component of our React application

- /css: contains .css files
- /constants: contains useful constants like the API urls and a DEBUG_MODE used during development
- /images: contains all the static images
- /utils: contains some functions used all over the application in order to not duplicate code: HTTP requests, parser for dates, an error pop up creator
- /components: contains all the main react components and another folder called /sub-components containing all the less important components.

4.C Database

/code/database contains the sql dump of the database.

5. Testing

We made two main types of tests: *unit tests* and *integration tests*.

5.A Unit Test

In order to perform our unit tests easily and correctly we adopted **Mockito,** which is a popular framework for writing JUnit software tests in java. We decided to use Mockito since most of the classes we needed to test have external dependencies on other classes and we desired to have a controlled environment in order to correctly validate the functionalities of our classes.

We performed either behavioural tests and functional tests.

We put particular effort in testing the business component of our project.

Here we show the results:

Module	classes	lines
QSB	100%	80%
AMB	100%	85%
SSB	100%	81%

Particular effort has been put in testing the TicketSchedulerComponent, which is a particularly delicate class.
This class contains the algorithm that arranges all the tickets of a shop in a queue. More details on this algorithm are available in the Design Document.

During the design phase of the algorithm, we built a static page that thanks to JavaScript created random data, ran the algorithm and printed the result in html, in order to have a visual confirmation that the algorithm was working.

In the same way, after we have developed the algorithm in Java in the server, in order to visualize the result, we have build a Queue Page in the frontend that is actually a visualizer for the output of the algorithm: the TicketSchedulerComponent will print in the console the time line in a json format and by copy and pasting the result on the Queue Page we'll be able to visualize the queue in a human-friendly way.

This entire process helped us resolve lots of bugs and to refine the algorithm to optimize it.

To test the algorithm we have considered edge cases and also tried to stress with heavy loads.

5.B Integration Testing

5.B.2 Integration Testing with the Data Layer

Since we followed a data-first approach to testing, we started by validating the Data Layer components. It took us little effort since we use queries with little parameters and with trivial output results. We basically largely took advantage of the SQLWorkBench tool in order to check whether our modules worked out properly. A main drawback of this approach to the integration with the data layer is that it took us much time and it's infrastructure dependent.

A better approach would have consisted in using the Spring Boot functionalities integrated with DBUnit library in order to save time and have many basic functionalities validated by default.

5.B.2 Integration Testing with the Web Modules

In order to integrate the web Module, we took advantage of the use of **Swagger**.

Thanks to this tool we have been able to stress our application with the help of several test sets.
Some of them have been thought with the aim to stress a single microservices and to validate its behaviour in different use cases, while other have the aim to test the behaviour of the application while more of one microservice are contemporary in use, either to carry out different tasks or to cooperate to bring off a common task.

Here are some of the test we have effectuated:

AMW/

userinfo

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized

login

- not existing username: 404, "no user found"
- wrong password: 401, "credentials provided are not correct"
- empty password or username: 400, empty credentials

registration

- missing or wrong username: 404, “no user found”
- already in use username: 406, “username already in use”
- the *inset password* and the *confirm password* fields don’t match: 400, “passwords don’t match”
- missing or wrong password: 401, “credentials provided are not correct”
- any field empty : 400, “credentials are empty”

logout

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized

SSW/

shops

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user isn't a manage: 401, unauthorized

AllShops

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user is a manager: 401, unauthorized

tickets

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user is a manager: 401, unauthorized

ticketDetail/{ticketid}

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user is a manager: 401, unauthorized
- if the username exists, such user is not a manager, but the ticket requested exists but it doesn't belong to the requiring user: 401, “not authorized”

shopDetail/{shopid}

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user is not a manager: 401, unauthorized
- if the username exists, such user is a manager, but the shop requested exists but it doesn't belong to the requiring user: 401, “not authorized”

shopShifts

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if the username exists, but such user is not a manager: 401, unauthorized
- if the user exits, is a manager, but they are not allowed to add shifts to such shop: 401, unauthorized
- if the the opening time is after the closing time or the day of the week inserted does not make any sense : 400, “corrupted data”

newShop

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if not a manager: 401, unauthorized
- if no image uploaded: 400, “upload an image”
- if inconsistent data in the shop specific (such as shop negative shop capacity etc) : 400, “inconsistent data”
- if inconsistent data in the shop specifics (such as shop negative shop capacity etc) : 400, “inconsistent data”
- if inconsistent data in the shop shiftspecifics (such as shop negative shop capacity etc) : 400, “new shop have been created but shop shifts couldnt be added because of inconsistent data”

QSW/

enqueue

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- user is a manager: 401, unauthorized
- inconsistent data (such as too long permanence time) : 400, specific message of explaining the inconsistency of data
- if user already has a ticket: 400, “you already have a ticket”

dequeue

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- user is a manager: 401, unauthorized
- user has no tickets: 400, “you have no tickets”
- ticket id doesn’t belong to the requesting user: 400, “not your ticket”

getQueueInfo

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if doesn’t exists a shop with such a shopid: 400, “no such shop”
- if queue of the requested shop is empty: 200, waiting time 0

scanTicket

- missing or wrong username header: 401, unauthorized
- missing or wrong session-token header: 401, unauthorized
- if user is a manager: 401, unauthorized
- if the ticket doesn’t belong to the user who is scanning it: 400, “not your ticket”
- if the ticket doesn’t exists: 400, “no such ticket”
- if the ticket status is “invalid”: 400, “ticket not valid”
- if the ticket has an exit time but not an entrance time: 400, “exit without entrance”

6. Installation instructions

6.A Server

[Instructions for a **windows machine**]

1. Install Java 15 (or higher).
2. Open a powershell window with admin privileges and navigate to the folder containing this file (**`cd repo_location\LermaMainettiGambera\delivery`**)
3. Go to /backend (**`cd backend`**) and run the clup.bat script (**`.\clup.bat`**).

   Now a lot of terminal should open, let them do (you can tell they have finished when the last word printed is “milliseconds”).
   At this point the server is running on your machine and should be reachable at the addresses:

   a. http://localhost:8082/
   b. http://localhost:8081/
   c. http://localhost:8089/

   Also the script should have started an SQL server at the port 3306

4. To see if the services have been correctly started run swagger.bat script (**`.\swagger.bat`**). It will open three swagger pages, one for each service, on your predefined browser.

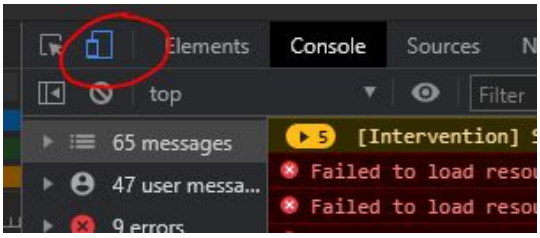
Please ensure that you have no other processes at localhost holding the ports: 8080, 8081, 8089, 8005, 8006, 8007, 8443, 8444.

Please be sure no other SQL service is running gon port 3306.

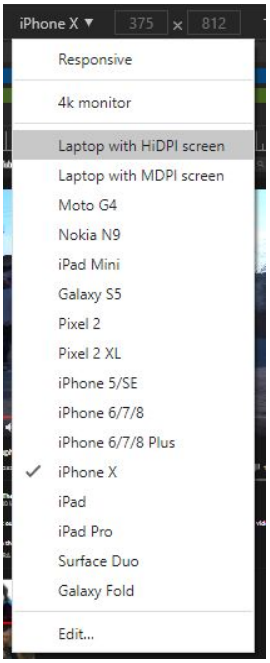
6.B Web app

1. Be sure to have npm and node.js installed.
2. Open a powershell window with admin privileges and navigate to the folder containing this file (**`cd repo_location\LermaMainettiGambera\delivery`**)

3. Go to /frontend (**`cd frontend`**) and run the downloadLiveServer.bat script (**`.\downloadLiveServer.bat`**). This script will run the npm install -g serve command, that install the serve package, a static file serving and directory listing.
4. In the same folder you now have to run the liveserver.bat script (**`.\liveserver.bat`**). This script will open a live server on http://localhost:5000/ (if the port 5000 is free, otherwise you can see in the console where the web app is hosted) that you can visit on the browser to see the React web app.
5. The web app was build for mobile, so we encourage to use it with smartphone screen sizes. There are two options:
    a. [for google chrome]: open the developer tools (three dots in the top right corner > other instruments > developer tool) and click on the "smartphone/tablet" icon:



Also from the drop down menu select the "iphone x" option:



    b. Just resize the browser window to be smartphone-like.

Effort spent

Federico Mainetti Gambera: ~150h

Ludovica Lerma: ~150h

Part of the effort was spent in choosing and learning how to use new technologies, setting up the development environment, and solving unexpected problems,

Division of the major tasks: Federico Mainetti Gambera took care of the front-end and the queue algorithm, while Ludovica Lerma took care of the back-end and the relative tests.

Despite this split, both members contributed equally in the decision making process and helped each other throughout the entire development.