POLITECNICO MILANO 1863

Implementation and Test Deliverable

Customers Line-up

Installer - Source code

Alessandro Ferrara - Lorenzo Fratus
Professor: Elisabetta di Nitto

February 05, 2021
Version: 1.0

Contents

1. Introduction **3**
A. Purpose . . . 3
B. Scope . . . 3
C. Definitions, acronyms and abbreviations . . . 3
C.1. Definitions . . . 3
C.2. Acronyms . . . 4
C.3. Abbreviations . . . 4
D. Revision history . . . 5
E. Reference documents . . . 5
F. Document structure . . . 5

2. Product functions **6**
A. Implemented functions . . . 6
A.1. Join a queue (digital) . . . 6
A.2. Join a queue (physical) . . . 6
A.3. Store overview . . . 6
A.4. Ticket scan . . . 7
B. Discarded functions and details . . . 7
C. Other changes . . . 7

3. System development frameworks **8**
A. Adopted frameworks . . . 8
B. Adopted programming languages . . . 8
B.1. JavaScript . . . 8
B.2. Other languages . . . 9
C. Adopted middlewares . . . 9
C.3. Other libraries . . . 10
E. Adopted APIs . . . 10
E.1. Discarded API . . . 10
E.2. Replacement API . . . 10

4. Source code structure **11**

5. Testing activity **12**
A. Testing environment 12
B. Test cases 13
B.1. Account routes 13
B.2. Clupper routes 13
B.3. Store manager routes 14
B.4. Middlewares 15
C. Additional features 15

6. Installation guide **16**

7. Effort spent **17**
Pair programming 17
Ferrara Alessandro 17
Fratus Lorenzo 17

1. Introduction

A. Purpose

The purpose of this document is to present the work done during the implementation of the first prototype of this application.

This document presents a summary of the functions that are available in the current version of the software and an overview of the languages and frameworks used during the development.

Finally, it outlines the tests that have been carried out and presents a simple installation guide.

Together with the RASD and the DD, this document has the purpose to inform about the realization of the software called Customers Line-up.

B. Scope

As better explained in the previous documents, the software wants to foster a safe shopping experience by providing a customer flow control system. This system should allow the users to avoid crowds both inside and outside the stores.

The system should be very simple to use (to adapt to all demographics) and provide fallback options for those who do not have access to the application.

C. Definitions, acronyms and abbreviations

C.1. Definitions

- **User**: person which is correctly registered to the system, it abstracts the concepts of clupper and store manager.
- **Customer**: person which wants to enter a store, it abstracts the concepts of clupper and guest.

- **Clupper**: person (both user and customer) which is able to use the basic services offered by CLup (join a queue and book a visit).
- **Guest**: person (customer) which is not registered to the system or that is currently unable to use the application, and therefore cannot take advantage of the services offered by CLup.
- **Store manager**: person (user) which is able to use the managerial services offered by CLup.
- **Store**: physical business registered to the system (by registering his store manager).
- **Store capacity**: maximum number of customers allowed inside a store (according to the regulations currently in force it corresponds to half of the capacity of the building).
- **Ticket**: QR code issued to a user as a receipt for queuing or booking at a store. It can be digital or physical and grants the access to the store based on the transaction that generated it.
- **Valid ticket**: a ticket is valid if the store contained in it matches the one of the store manager which scanned it and:
  - the ticket is the first in the store queue, if the ticket has been generated by a "Join a queue" function;
  - the time slots linked to the ticket include the current timestamp, if the ticket has been generated by a "Book a visit" function;
- **Queue**: imaginary list of tickets, bound to a store, ordered by their time of issue, it does not contain tickets of customers with a reservation.

C.2. Acronyms

- **RASD**: Requirement Analysis and Specification Document.
- **DD**: Design Document.
- **ITD**: Implementation and Testing Deliverable.
- **GPS**: Global Positioning System.

C.3. Abbreviations

- **CLup**: Customers Line-up.

D. Revision history

Version	Date	Description
1.0	05 Feb 2021	First version

E. Reference documents

- Assignment document A.Y. 2020/2021 ("Requirement Engineering and Design Project: goal, schedule, and rules")
- Assignment document A.Y. 2020/2021 ("I&T assignment goal, schedule, and rules")
- Requirement Analysis and Specification Document - Customers Line-up, version 1.1 (referred to as "RASD" in the document)
- Design Document - Customers Line-up, version 1.1 (referred to as "DD" in the document)

F. Document structure

- **Section 1: Introduction**: The first section provides an introduction to the purpose of the document and the scope of the CLup system. Included here is a glossary with the definitions, acronyms and abbreviations used in this document.
- **Section 2: Product functions**: This section is a reference to the previous documents. Here are presented the functions that are actually implemented in the software and those that are not (together with the motivation for excluding them).
- **Section 3: System development frameworks**: In this section are explained the adopted development framework which also includes an overview of the adopted programming languages, middlewares and APIs.
- **Section 4: Source code structure**: This section contains the structure of the source code, with some information about the content of the main folders.
- **Section 5: Testing activity**: Here is specified how the system have been tested and are described the main test cases and their outcome.
- **Section 6: Installation guide**: This section includes a simple installation guide.

2. Product functions

A. Implemented functions

A.1. Join a queue (digital)

This function allows a clupper to line up for the desired store without having to immediately reach the store. After selecting a supermarket from the list the clupper will be able to see its details (name, address, number of customers already in line) and to join the queue. The system will provide the clupper with a digital ticket, which he will be able to see directly from the application. Finally, the clupper is able to leave the queue at any time before entering the store, this results in the deletion of his ticket.

A.2. Join a queue (physical)

This function is a fallback of the first one, indeed, it allows the store manager to insert into the queue any guest requesting it. The system will provide the store manager with a digital ticket, which he will have to convert into physical (by printing it) and hand out to the guest. Finally, the guest is able to leave the queue at any time before entering the store, by asking the store manager to delete his ticket.

A.3. Store overview

This function allows a store manager to have a live overview of the store. The system will provide the store manager with the number of customers inside the store (compared to the capacity) and those in the store queue. This data should allow him to regulate the flow of customers. This function also offers to the store manager the possibility to update the store capacity in order to adapt to the latest regulations.

A.4. Ticket scan

This function allows a store manager to scan (and validate) a customer's ticket (both at the entrance and the exit of the store). This has the double purpose of updating the data contained into the store overview, and make sure that the flow of customers corresponds to that expected by the system (no one is able to enter before his turn).

B. Discarded functions and details

This is a list of functions or functional details that are described in the previous documents but have been excluded from the implementation of the system, mostly due to its prototypical nature.

- **Book a visit function**: this advanced functionality was not required by the assignment document.
- **Opening time of stores**: this detail was directly connected with the implementation of the *Book a visit* function, it should have been used to calculate the available time slots for each store.
- **User notifications**: the option to notify the clupper to approach the store requires to monitor his position, this was not possible with the adopted API.
- **Map exploration and search by address**: this minor functionalities have been discarded because not available with the adopted API (more on that in the next chapter).

C. Other changes

Differently from what shown in the DD:

- **On the web tier**: there is no *load balancer* to distribute the load between multiple instances of the application tier.
- **On the application tier**: there is no *PM2 process manager*, a docker-like process manager to spawn multiple instances.

These modules are not mission-critical and have been avoided in order to focus on other (more important) aspects of the system.

3. System development frameworks

A. Adopted frameworks

To facilitate the development process, the application has been built starting from already existing frameworks.

- **Express**: as already shown in the DD, the system business logic resides on a server running on NodeJS. ExpressJS is considered the *de facto* web framework for this type of execution environment.
- **Mocha**: is the testing framework that provides all the functions and interfaces used to test and benchmark the software.

B. Adopted programming languages

B.1. JavaScript

The whole logic of the system has been written using JavaScript ES6, an high-level just-in-time compiled programming language that, together with HTML and CSS, is one of the core technologies of the WWW.
In this application, the logic is almost completely on the web and application servers, with the workload on the client side only responsible for making the page interactive.

B.1.1. JavaScript Pros

- **Weakly Typed**: being weakly typed allows a lot of flexibility that let us develop faster and change things quickly.
- **I/O driven**: NodeJS operates on a event loop which means, thanks to the asynchronous behavior that it can handle thousands of concurrent connections.

- **Widely used**: it's the most used programming language, which means it is heavily documented and makes the system highly maintainable.

B.1.2. JavaScript Cons

- **Weakly Typed**: a weakly typed programming languages is prone to errors.
- **Callback Hell**: due to the asynchronous nature of JS, situations in which there are multiple levels of nested callbacks, can affect the system performance very easily.

B.2. Other languages

In addition to JS, HTML and CSS has been used to build the structure and appearance of each web page.

C. Adopted middlewares

Here a list of the middlewares that have been used to add functionalities to the application server.

- **Cookie Parser**: used to parse cookies contained in the incoming requests.
- **Express Session**: used to manage user sessions on the application tier.
- **Pug**: template engine, used to perform server-side rendering of HTML pages.

Generally speaking, using a middleware has several advantages including:

- **Already developed**: there is no need to reinvent the wheel, using an already developed module to provide a standard function saves time and resources.
- **Field-tested**: external modules are commonly used in development, this typically ensures that it is well functioning and that any new feature is tested by a lot of developers.

The only drawback compared to a custom-made middleware is that the developer has to adapt to the interfaces offered by an external module.
However, with JavaScript being a widely used language, there is plenty of alternatives available.

C.3. Other libraries

This is a list of external libraries that are not middlewares but have played a key role in the implementation of this system.

- **Request**: used to allow HTTP request to the APIs.
- **MySQL**: used to perform queries on the DB.
- **QrCode**: used to encode ticket ID's into png images.
- **Haversine Distance**: used to calculate the distance between two GPS locations.
- **Superagent**: testing, simulates an HTTP client.
- **Chai**: testing, provides an assertion library.

E. Adopted APIs

E.1. Discarded API

Unlike what is suggested in the Design Document, the implemented system does not relies on *Google Maps Javascript API* to provide GPS-related functions.
Even if it is clearly the best solution to build the definitive version of this application, this API requires a lot of time to be mastered to have access to all its functionalities.
Being this the implementation of a prototype, the main concern has been to build a version of the application that is able to provide the most important features.

E.2. Replacement API

To replace the discarded Maps API, a simpler but equally reliable service has been used: PositionStack API.
This API enables our application to perform a conversion from a GPS position (expressed by means of latitude and longitude) to an address and back.
This conversion involves minimal error (that can lead to a wrong civic number) that was deemed acceptable in favor of a faster implementation.
Moreover, due to the way in which this API is implemented, the conversion is possible only by region (in this case only *Lombardia*). This limit could be canceled in the future for example by asking the user to select his region in the registration process.

4. Source code structure

This is a simple directory tree that shows the structure of our source code.

```
/
└── src
│   ├── model
│   │   Model of the system, to enable the usage of the proxy pattern
│   ├── view
│   │   ├── css
│   │   │   CSS files to define the aspect of the pages
│   │   ├── js
│   │   │   Client-side JS code used in some pages
│   │   ├── img
│   │   │   Static images used in some pages
│   │   ├── include
│   │   │   Static imports shared through all the pages
│   │   └── template
│   │       Pug templates for HTML pages
│   ├── controller
│   │   ├── database
│   │   │   Controller that handles DB connections with a connection pool
│   │   ├── middlewares
│   │   │   Custom middlewares that handle authentication and authorization
│   │   ├── routes
│   │   │   Routes (APIs) provided by the application
│   │   ├── services
│   │   │   Exposed services accessed by the routes to carry out business logic
│   │   └── server.js
│   │       Initializes the routes and the middlewares
│   └── index.js
│       Main method that runs the server
└── test
    Tests performed on the system
```

5. Testing activity

A. Testing environment

To ensure a testing activity as close as possible to a real-world scenario, the test suite was run directly using the production database rather than a custom-made stub.

In order to stimulate the system, a mock HTTP client was introduced thanks to an external module. This client has the task of performing multiple actions on the available services and verifying that the response matches the expected result for each test.

Due to the small size of the public interface classes, to avoid trivial tests, the test cases focus specifically on the routes of the web server. These components directly interact with the public interfaces, in this way, a complete test suite is guaranteed.

The objective of this activity is to cover the requirements highlighted in the RASD in order to ensure that the system is able to meet them. The test cases are grouped according to the public interface they refer to.

B. Test cases

B.1. Account routes

The testing of the account routes begins with the initialization of the CLup application and the instantiation of a HTTP client to perform the testing.

The following tests are done:

1. Login:
    - Login of a clupper
    - Login of a store manager
    - Login with invalid credentials
2. Registration of a clupper:
    - Registration with valid information
    - Registration with missing information
    - Registration with email already in use
3. Registration of a store manager:
    - Registration with valid information
    - Registration with missing information
    - Registration with email already in use
    - Registration with VAT already in use

All the test are passed. The benchmark results are under 1000ms.

B.2. Clupper routes

As in the account routes testing, the procedure initializes both CLup application and a HTTP client. All the tests are carried out sequentially (the first one is always the login).

The following tests are done:

1. Store retrieval:
    - See the list of stores
    - See the details of a store
    - See the details of an invalid store

2. Queue management:
    - See the details of the queue ticket
    - See the details of the queue ticket when the clupper is not in a queue
    - Join the queue of a store
    - Join the queue of an invalid store
    - Leave the queue of a store
    - Leave the queue of an invalid store

All the test are passed. The benchmark results are under 600ms.

B.3. Store manager routes

The testing of the store manager routes proceeds in the same way as the testing of the clupper ones (always starting from the login).

The following tests are done:

1. Capacity update:
    - Update the capacity with a valid value
    - Update the capacity with an invalid value
2. Ticket scan:
    - Scan at the entrance
    - Scan at the exit
    - Scan of a non-existing ticket
    - Scan of a ticket that is not the head of the queue
    - Scan of a ticket when maximum capacity is reached
3. Other ticket operations:
    - See the list of issued tickets
    - Issue a new ticket
    - Delete an issued ticket

All the test are passed. The benchmark results are under 600ms.

B.4. Middlewares

Some small tests are also performed to ensure that the middlewares are activated at the right time.

The following tests are done:

1. No login:
   - Access explore page without login
   - Access overview page without login
2. Clupper:
   - Access explore page with login
   - Access overview page with login
3. Store manager:
   - Access explore page with login
   - Access overview page with login

All the test are passed. The benchmark results are under 600ms.

C. Additional features

Together with each test case, a benchmark is automatically run to measure the performance of the system in terms of time required to complete the tasks. The result of the benchmark could vary a lot depending on the network speed and hardware it is running on.

6. Installation guide

A detailed installation guide, complete with images, can be found in the *README* file contained both in the installation folder and in the GitHub repository at this link.

7. Effort spent

Pair programming

Topic	Hours
Sections 2, 3 and 4	3.0h

Ferrara Alessandro

Topic	Hours
Installation guide	0.5h
Section 5	1.5h

Fratus Lorenzo

Topic	Hours
Section 1	1.0h
Section 5 revision	1.0h