



**POLITECNICO**  
**MILANO 1863**

## **CLup - Customer Line Up**

*Software Engineering 2 Project 2020/21*

### **Authors**

- Francesco Attorre - *10618456*
- Thomas Jean Bernard Bonenfant - *10597564*
- Veronica Cardigliano - *10627267*

---

<b>Deliverable:</b>	DD
<b>Title:</b>	Design Document
<b>Authors:</b>	Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano
<b>Version:</b>	1.0
<b>Date:</b>	10-January-2021
<b>Download page:</b>	<a href="https://github.com/FrancescoAttorre/softeng2-attorre-bonenfant-cardigliano">https://github.com/FrancescoAttorre/softeng2-attorre-bonenfant-cardigliano</a>
<b>Copyright:</b>	Copyright © 2021, Francesco Attorre, Thomas Jean Bernard Bonenfant, Veronica Cardigliano - All rights reserved

---

## Contents

<b>Table of Contents</b>	<b>3</b>
<b>1 Introduction</b>	<b>4</b>
A Purpose	4
B Scope	4
C Definitions, Acronyms, Abbreviations	4
C.1 Definitions	4
C.2 Acronyms	4
C.3 Abbreviations	4
D Revision History	4
E Reference Documents	4
F Document Structure	4
<b>2 Architectural Design</b>	<b>6</b>
A Overview: High-level components and their interaction	6
B Component view	6
C Deployment view	7
D Runtime view	7
D.1 oneSubsectionPerAction	9
E Component Interfaces	9
F Selected architectural styles and patterns	9
G Other design decisions	10
<b>3 User Interface Design</b>	<b>11</b>
A UserMobileApp Interfaces	11
B WebApp Interfaces	18
<b>4 Requirements Traceability</b>	<b>19</b>
A External Interface Requirements	19
<b>5 Implementation, Integration and Test Plan</b>	<b>21</b>
A Implementation	21
B Integration	21
C Test plan	21
<b>6 Effort Spent</b>	<b>22</b>
<b>7 Used Tools</b>	<b>23</b>

# 1 Introduction

## A Purpose

The goal of this Design Document is to provide a general view of the architecture of the software, adding more technical details to the information provided in the RASD document. The architecture will be explained in terms of hardware and software components, the interfaces they provide and the interactions between them. The document is going to face also the integration and testing plans, and the main design patterns to be exploited. Information related too much on the implementation will not be dealt in detail since this document aims to the description of an high level architecture of the system.

## B Scope

CLup is an application which aims to manage crowds in a period of global pandemic. It allows store managers to monitor the entrances in a building and to avoid the the formation of queues in front of stores. CLup will allow people to save time and be safer, managing both the accesses of people who use the application directly, and of people who go physically to the store, using store managers as intermediaries. CLup can be accessed by customers, store managers and activities which will register stores available to be visited. Customers using the application, moreover, will be notified when they're supposed to leave for the store, and, when registered, they will have the possibility to book a visit to a store, being informed on alternative time slots or similar stores when nothing is available for their choices. Booking a visit, customers can decide how much time they want spend in the building and, optionally, which specific departments they want to visit, in order to allow a greater number of people to enter when possible.

## C Definitions, Acronyms, Abbreviations

### C.1 Definitions

### C.2 Acronyms

- **RASD**: Requirements Analysis and Specification Document
- **GPS**: Global Positioning System
- **CLup**: Customers Line-up
- **DD**: Design Document

### C.3 Abbreviations

- **Rn**: requirement number n

## D Revision History

## E Reference Documents

- **Specification Document**: "R&DD Assignment AY 2020-2021.pdf"
- **Slides of the lectures**

## F Document Structure

This DD is composed by 7 main sections:

- SECTION 1 is the introduction ...

- SECTION 2 contains the architectural design
- SECTION 3 with the user interface design
- SECTION 4 contains requirements traceability showing how the requirements described in the RASD map the design components identified in this document.
- SECTION 5 concerns the implementation, integration and testing. Here it is defined how the subcomponents should be implemented and integrated and which kinds of tests should be carried out on them.
- SECTION 6 contains a table with the effort spent by each member of the group.
- SECTION 7 references/tools.

## 2 Architectural Design

### A Overview: High-level components and their interaction

### B Component view

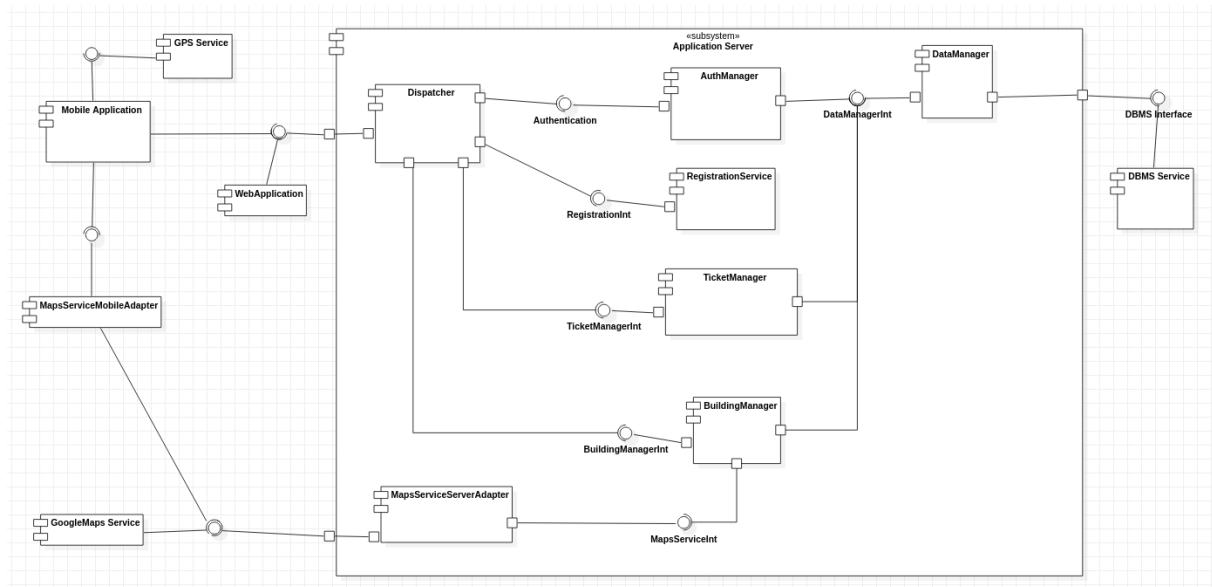


Figure 1: Component Diagram

#### **TicketManager**

This component provides services to deliver new tickets, change their state and communicates with BuildingManager to associate them with a Building.

#### **AuthManager**

This component offers services to authenticate users and deliver them tokens that authorize to use the other services. The dispatcher component will ask the Auth Manager to check the token validity before forwarding requests to other Components.

#### **BuildingManager**

Provides services to add buildings to the system, track the number of people in specific buildings, append and pop line up tickets from queues and add booking tickets to a building.

#### **RegistrationService**

Allows UnregisteredAppCustomers and Activities to register to CLup through its service.

#### **GoogleMapsService**

External service component that provides web mapping. Used to locate Buildings nearby.

#### **Dispatcher**

This component manages the requests received from MobileApp and Web Server components and redirects each one to the correct component able to handle it.

#### **Mobile Application**

Acts as a Client sending requests to the Dispatcher component. It is a fat client since it does contain

Applicative logic.

### **Web Application**

Web Client sending requests to the Dispatcher like the Mobile Application. It will be thinner than the Mobile Application taking part only at the Presentation Level since it does not contain any Applicative logic.

### **DataManager**

Manages datastructures and objects to persist to the database.

### **DBMS Service**

Represents the DBMS and the service he offers. It communicates with the DataManager

### **GPS Service**

Positioning system used to locate a device.

### **MapsServiceMobileAdapter**

Adapter that offers to the MobileApplication component an interface for using Mapping services. Useful if the external Mapping service changes the Mobile Application will always use the same interface.

### **MapsServiceServerAdapter**

Like previous adapter but for the BuildingManager component. Having 2 different Adapters offers the possibility to use different Mapping services for MobileApplication and BuildingManager

## **C Deployment view**

## **D Runtime view**

Here are proposed sequence diagrams in order to describe the way components interact with each other to accomplish specific tasks (the ones shown in the use cases of the RASD document).

### Premise

It's not explicitly stated in every sequence diagram that a token checking is performed for every request done from the Client to the Server. In order to do this, the dispatcher before processing the request just received will perform on AuthServer a token validity check. Instead the identification action performed in some diagrams has the task to retrieve the User information.

### Authentication Sequence Diagram

Two different authentication procedures can occur, the first is an authentication on a RegisteredAppCustomer account, instead a daily access to CLup services can be requested by UnregisteredAppCustomer.

#### Description

Both of the following process lead to gaining a token, that is a temporary access key for Server functionalities. The second one is Daily in the sense that every operation on server would stay for at most one day. account based - an authentication request is made by the client, after correctly forwarding this request to AuthManager an account check is done. If the user exists and has no token already associated with its account, then a new one will be created. Client receives a response containing his token. daily - a daily authentication request is made by the client, after correctly forwarding this request to AuthManager an access check is done. If the client can get a token in order to access server basic functions a new association between a specific client and a new token is made, else a negative response is sent back to the client.

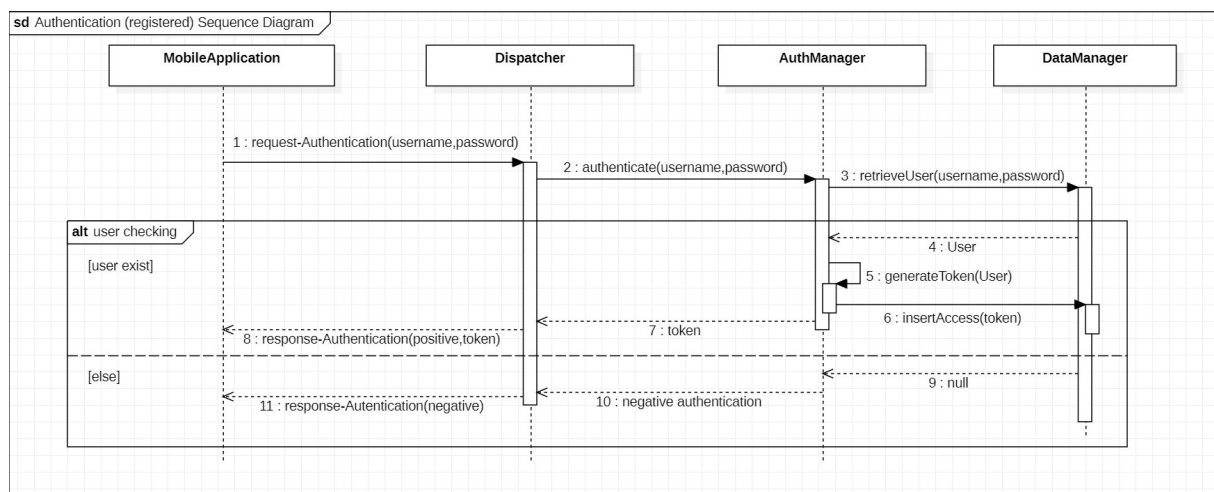


Figure 2: Authentication (registered) Sequence Diagram

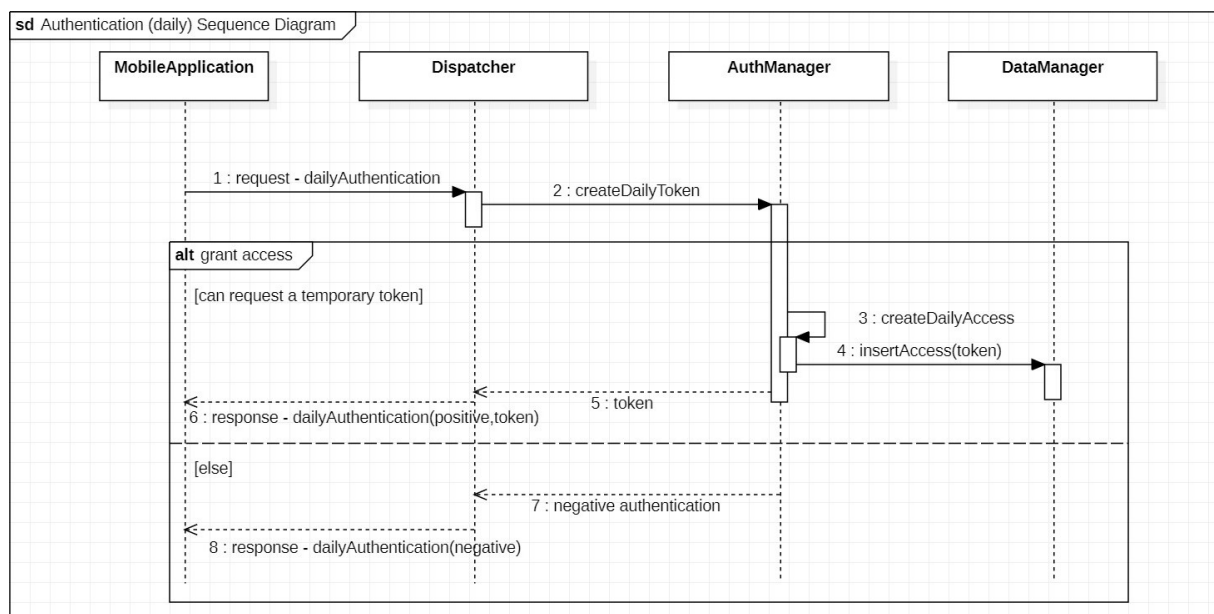


Figure 3: Authentication (daily) Sequence Diagram



### Acquire a LineUpDigitalTicket Sequence Diagram

*This diagram explains how a LineUpDigitalTicket can be acquired by an AppCustomer.*

#### Description

Client retrieves its position and sends a request in order to receive a list of buildings from which one would be selected. Building retrieval is effectively done by the DataManager. Whereas the MapsServiceAdapter (communication with a maps API) provide useful information to correctly pick only reachable buildings. Once the building is selected, Client sends another request in order to gain a ticket. An identification is performed on AuthManager to correctly retrieve the UserID (associated to the token) needed in ticket acquisition. Afterwards, to complete the process, if the building has at least one void space the ticket state is set to Valid, else the ticket is inserted in the queue. Note that only after a Discovery request from the client, this ticket will produce a notification (as described in next diagram).

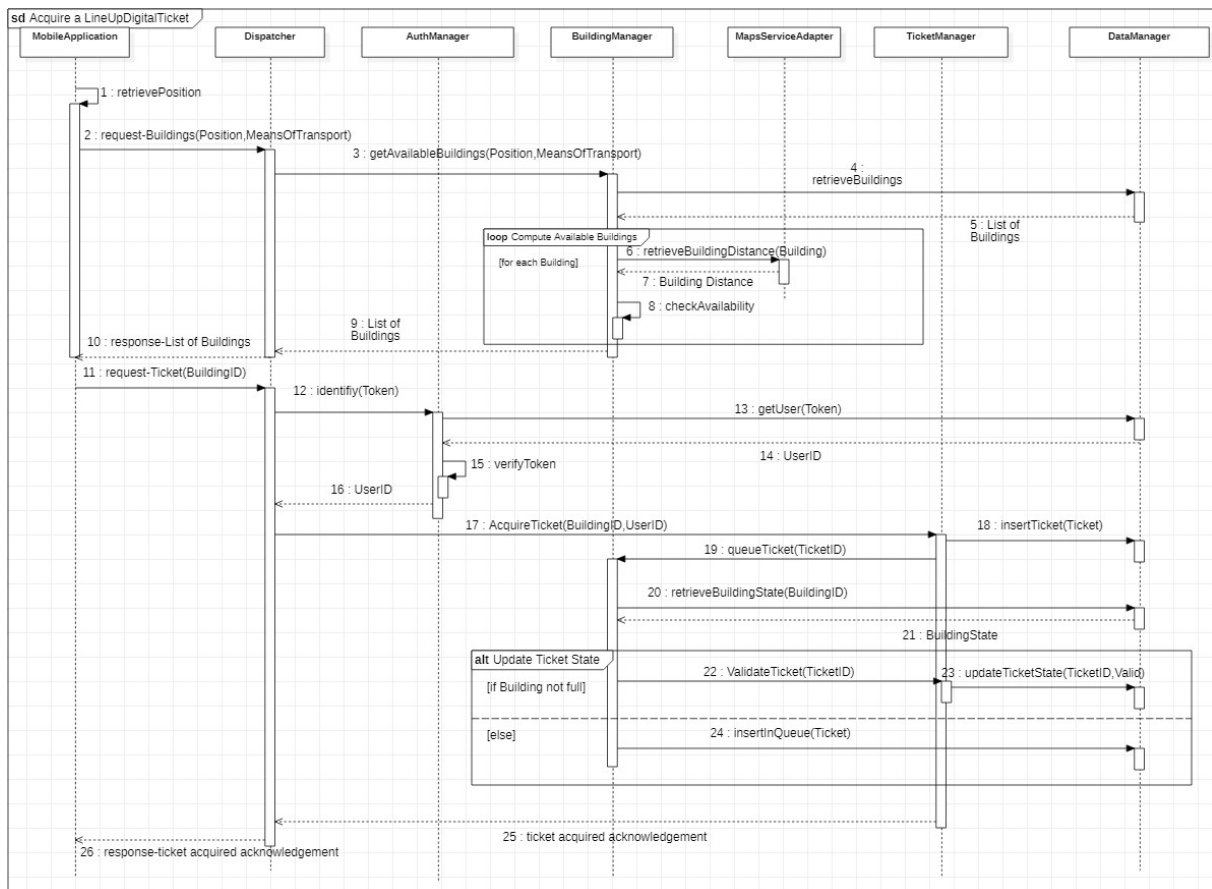


Figure 4: Acquire a LineUpDigitalTicket Sequence Diagram

#### D.1 oneSubsectionPerAction

### E Component Interfaces

The following diagram shows all the component interfaces already exploited in the sequence diagrams together with the dependencies between the various components.

### F Selected architectural styles and patterns

The architectural style selected is a three-tier client-server architecture, in order to have a good decoupling of logic, data and presentation, increasing reusability, flexibility and scalability. Moreover, compo-

nents in the application server have to be developed with low coupling among modules in order to make the system more comprehensible and maintainable. About the components, they have been designed to maintain a stateless logic as much as possible, that is, they should not contain an internal state, but refer to the database to get the necessary information about the system. This is important since instances of components can fail, go down and nothing must go lost in this eventuality. In this context the scalability of the database is so important, and `DataManager` will play a leading role. The protocol used to send requests is HTTP, which is a good choice to implement a RESTful architecture to meet the above objectives of having a stateless and low coupling system. Other advantages are that it would be cacheable and with a uniform interface. Data are transmitted in JSON, which is one of the simplest and most easily customizable protocols. It is also easy readable and allows fast parsing.

Finally, we've decided to use some design patterns in order to exploit existing models to solve recurrent problems. This benefits the reusability and maintainability of the code, as well as making it easier for designers to understand how the system works. Below the patterns used:

#### Model View Controller - MVC

MVC is a widely used pattern, particularly suitable for the development of applications written in object-oriented programming languages such as java. MVC is based on three main roles which are: the Model that contains all the methods to access the data useful to the application, the View that visualizes the data contained in the model and deals with the interaction with users and the Controller which receives the commands of the user and executes them modifying the other two components. In CLup, the Model logic is in ..., the controller part .... Instead, the view is represented mainly by ...

#### Adapter pattern

Adapter is a structural pattern that aims to match interfaces of different classes. The interface of the Adapter is interposed between the system and the Adaptee, that is the object to be adapted. In this way, whoever has to use a method of the Adaptee sees only an interface (or an abstract class) which would be implemented according to the component to be adapted. In the case of CLup, the component `MapsServiceAdapter` acts as the adapter, and `GoogleMapsService` is the component to be adapted. In this way, even assuming that the external service is changed, the internal system will not undergo any changes, since the new API will be handled by the `MapsServiceAdapter` component.

#### Facade pattern

Facade is also a structural pattern, which consists in a single class representing the entire subsystem. In the case of CLup, the `Dispatcher` takes all the requests from the client and then directs them to the specific component of the `AppServer`. The aim of this component is to mask the complexity of the entire subsystem, with which you can communicate via a simple interface.

## G Other design decisions

Thin/Fat client? DB?

### 3 User Interface Design

#### A UserMobileApp Interfaces

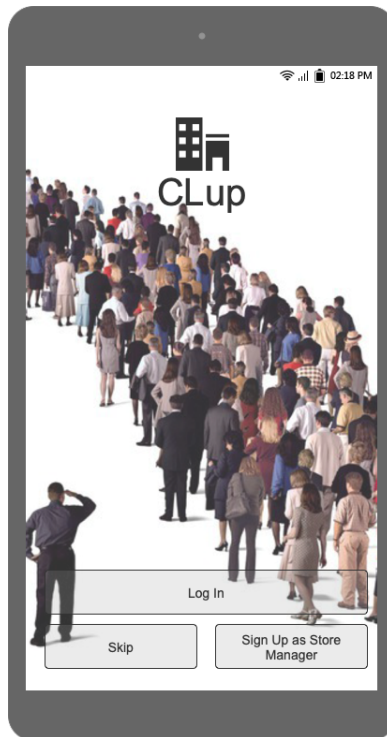


Figure 5: Starting page

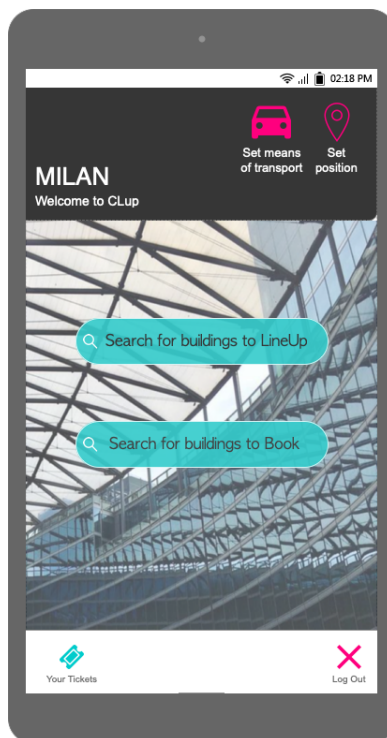
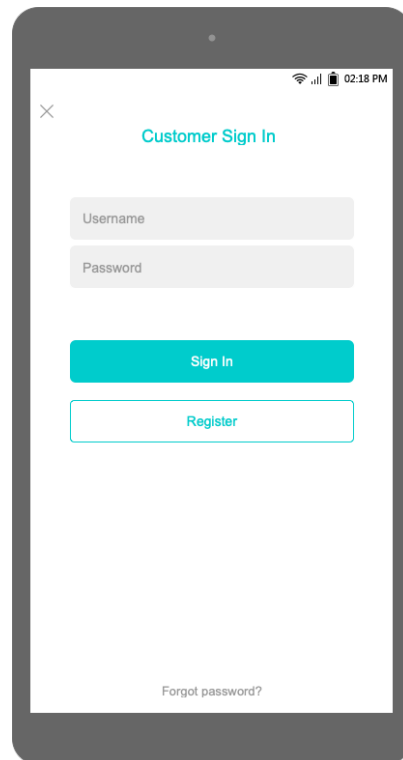
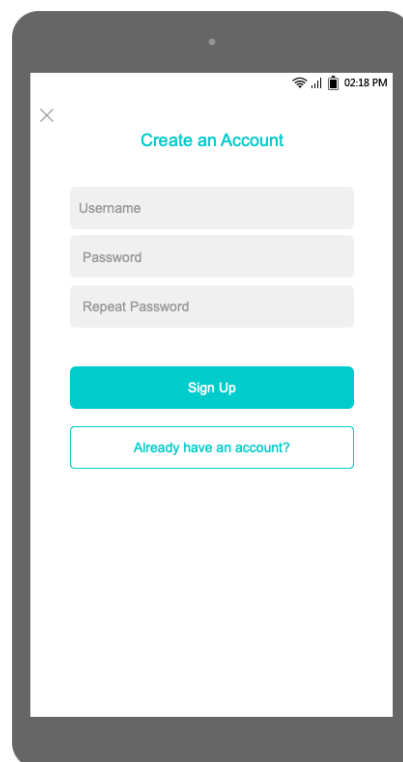


Figure 6: General home page of a registered customer



A mobile app interface for customer sign-in. The screen is white with a dark grey border. At the top right, there is a status bar showing signal strength, battery level, and the time 02:18 PM. Below the status bar is a close button (X). The title "Customer Sign In" is centered in teal. There are two input fields: "Username" and "Password", both with light grey backgrounds. Below these fields is a teal "Sign In" button. Underneath the button is a white "Register" button with a teal border. At the bottom, there is a link "Forgot password?" in small grey text.

Figure 7: Interface for customers that want to sign in



A mobile app interface for creating a new account. The screen is white with a dark grey border. At the top right, there is a status bar showing signal strength, battery level, and the time 02:18 PM. Below the status bar is a close button (X). The title "Create an Account" is centered in teal. There are three input fields: "Username", "Password", and "Repeat Password", all with light grey backgrounds. Below these fields is a teal "Sign Up" button. Underneath the button is a white "Already have an account?" button with a teal border.

Figure 8: Interface for customers that want to register to CLup

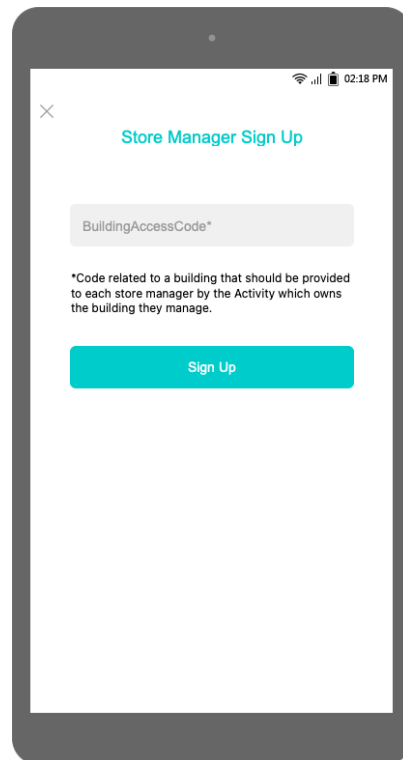


Figure 9: Interface for store managers to sign up



Figure 10: Home Page for store managers that have signed up

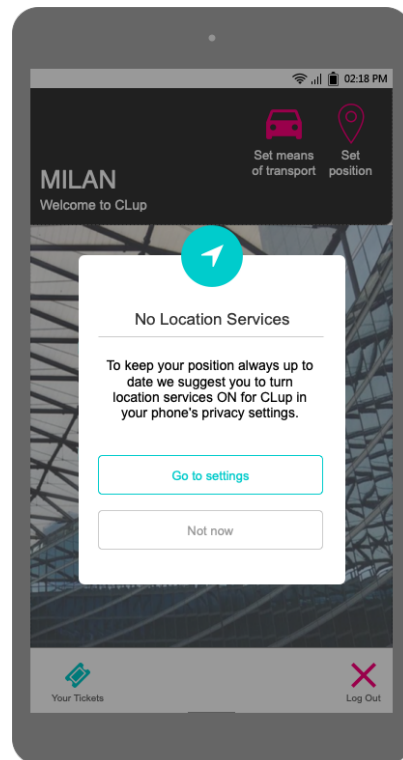


Figure 11: Pop-up to activate GPS services

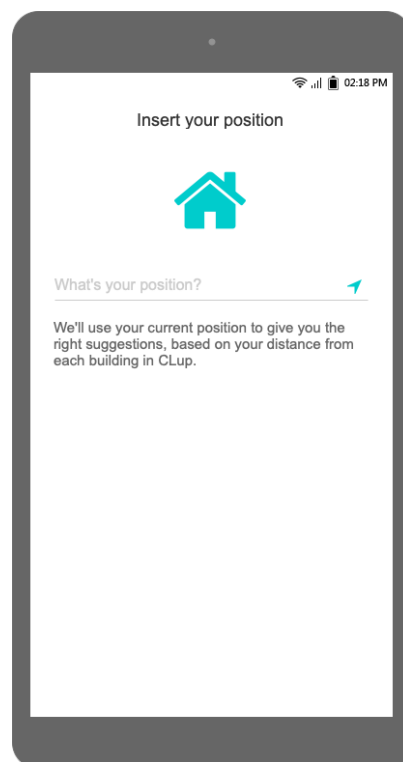


Figure 12: Setting page to set a position manually

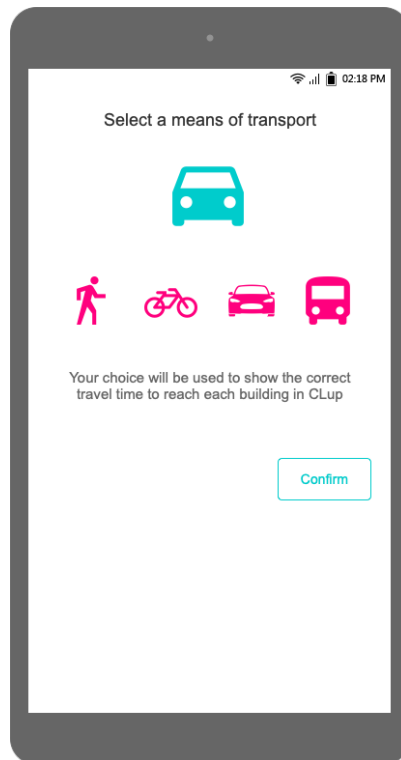


Figure 13: Setting page to choose a means of transport

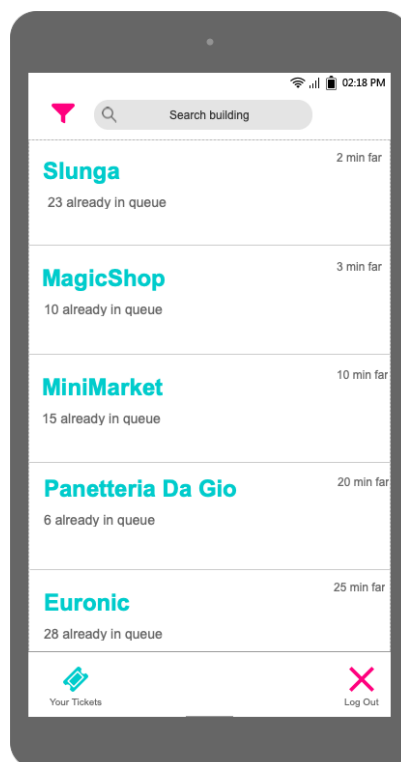


Figure 14: List of available buildings

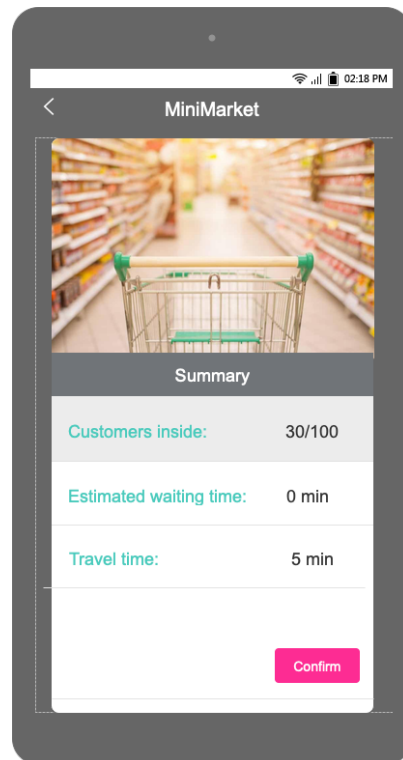


Figure 15: Summary page after the choice of a building to line up for.

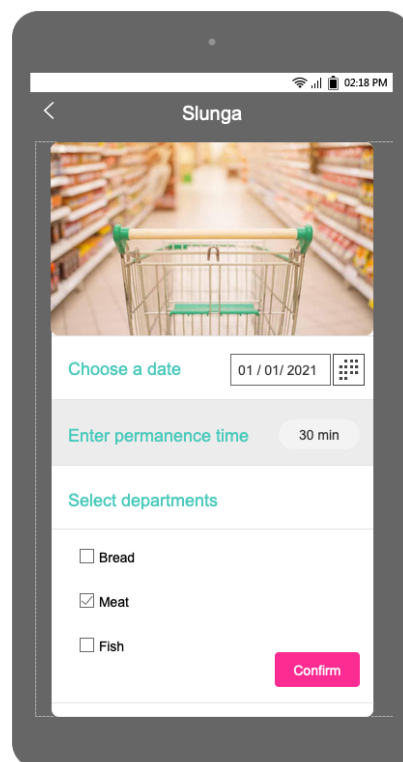


Figure 16: Options to set in order to book a time slot for the chosen building



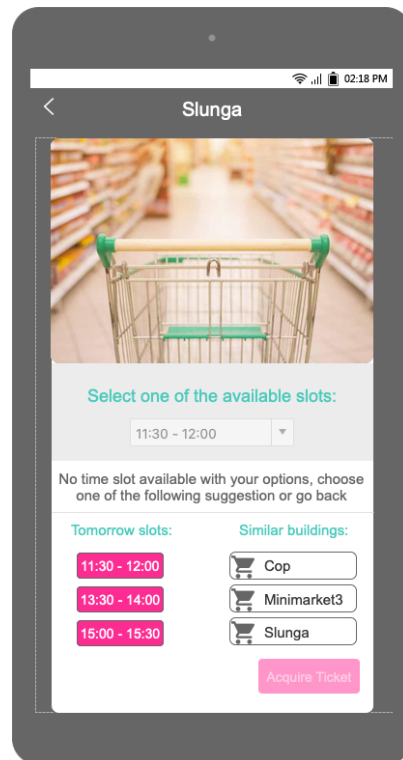


Figure 17: Interface to decide one of the available time slot/see alternative suggestions



Figure 18: List of tickets of a customers, with the possibility to view the related QR code

## B WebApp Interfaces

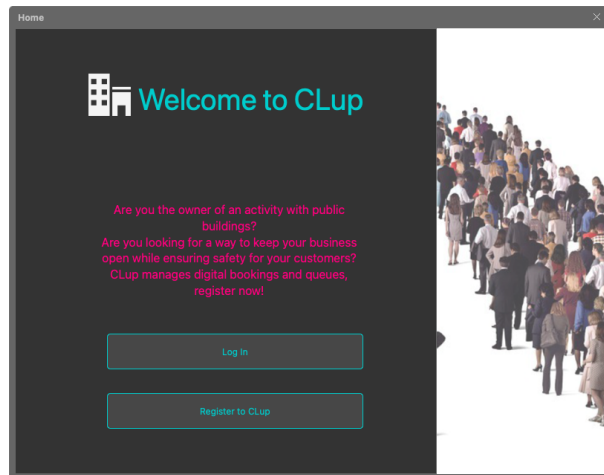


Figure 19: List of tickets of a customers, with the possibility to view the related QR code

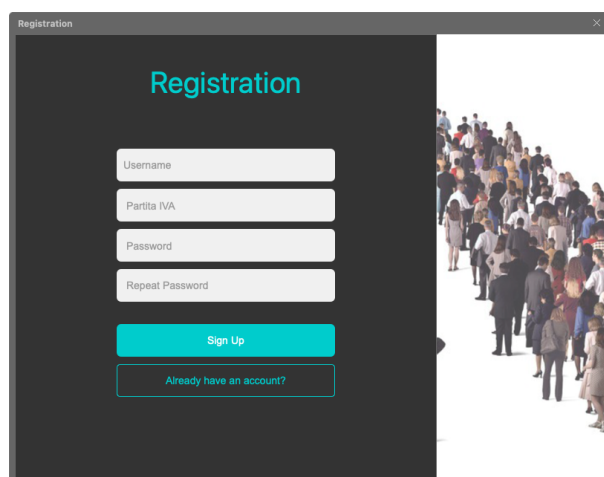


Figure 20: List of tickets of a customers, with the possibility to view the related QR code

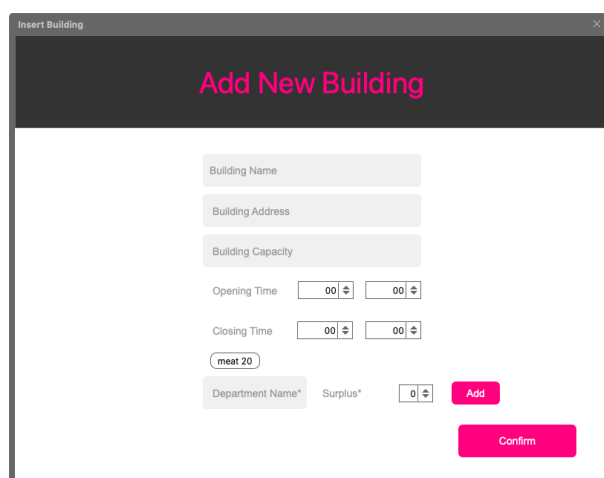


Figure 21: List of tickets of a customers, with the possibility to view the related QR code

## 4 Requirements Traceability

### A External Interface Requirements

-	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
R1						X								
R2						X								
R3				X										
R4				X	X									
R5								X						
R6								X						
R7								X						
R8								X						
R9								X		X				
R10										X				
R11										X				
R12											X			
R13											X			
R14											X			
R15											X			
R16											X	X		
R17					X									
R18							X							
R19							X							
R20							X							
R21							X							
R22							X							
R23									X					
R24							X							
R25							X							
R26							X							
R27							X							
R28													X	
R30									X					
R31														X
R32														X
R33														X
R35									X					

-	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
R36	X													
R37		X												
R38		X												
R39		X												
R40		X												
R41		X												
R42		X												
R43		X												
R44		X												
R45		X												
R46		X												
R47		X												
R48		X												
R49		X												
R50		X												
R51		X												
R52		X												
R53		X												
R54		X												
R55		X												
R56		X												
R57			X											
R58			X											
R59			X											
R60			X											
R61			X											
R62			X											
R63			X											
R64			X											
R65			X											
R66			X											
R68							X							
R69	X													
R70	X													
R71	X													
R72									X					
R73									X					
R74			X											
R76			X											
R77									X					
R78									X					

## **5 Implementation, Integration and Test Plan**

**A Implementation**

**B Integration**

**C Test plan**

## 6 Effort Spent

<i>Task</i>	<i>Name</i>	<i>Time spent</i>
x	Francesco Attorre	h
x	Thomas Jean Bernard Bonenfant	h
x	Veronica Cardigliano	h
x	Francesco Attorre	h
x	Veronica Cardigliano	h
x	Veronica Cardigliano	h
x	Francesco Attorre	h
x	Thomas Jean Bernard Bonenfant	h
x	Veronica Cardigliano	h
x	Veronica Cardigliano	h
x	Thoma Jean Bernard Bonenfant	h

comments

## 7 Used Tools

Tools used to create this RASD document:

- StarUML: for all the UML diagrams
- LaTeX: to create the pdf
- GitHub: for the repo of the project
- GoogleDoc: for a shared editing of the document