POLITECNICO
MILANO 1863

Design Document

Customers Line-up

Alessandro Ferrara - Lorenzo Fratus
Professor: Elisabetta di Nitto

January 28, 2021
Version: 1.1

Contents

1. Introduction

A. Purpose

The purpose of this document is to provide an overall guidance to the architecture of this software product and therefore it is primarily addressed to the development team.

More precisely, the document presents an overview of the high level architecture (defining the main components and their interaction) and additional details on the runtime behaviour and the user interface of the application.

Finally, it includes a plan for the implementation, integration and testing activity.

Together with the RASD, this document has the purpose to guide the developers in the realization of the software called Customers Line-up.

B. Scope

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.
To do so, the system offers to the users the possibility to:

- Join (and leave) a store queue.
- Book (and cancel) a visit to a store.
- Have an overview of the store.
- Manage the flow of customers by scanning their tickets.

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

- **Visitor**: person which is not registered to the system but is performing the operations to register.
- **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;
- **Booking/reservation**: digital ticket owned by a clupper to enter a store for one or more specific time slots.
- **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.
- **Time slot**: a half-hour time window.
- **Free time slot**: a time slot is considered free if the store is open during that period and the number of reservations already acquired over that slot does not exceed a quarter of the capacity of the store.

C.2. Acronyms

- **RASD**: Requirement Analysis and Specification Document.
- **DD**: Design Document.
- **GPS**: Global Positioning System.
- **JSON**: JavaScript Object Notation
- **REST**: Representational State Transfer

C.3. Abbreviations

- **CLup**: Customers Line-up.
- **Gx**: Goal number x.
- **Dx**: Domain assumption number x.
- **Rx**: Functional requirement number x.

D. Revision history

Version	Date	Description
1.0	03 Jan 2021	First version
1.1	28 Jan 2021	Adapted to RASD update

E. Reference documents

- Assignment document A.Y. 2020/2021 ("Requirement Engineering and Design Project: goal, schedule, and rules")
- Requirement Analysis and Specification Document - Customers Line-up, version 1.1 (referred to as "RASD" 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: Architectural design**: This is the core section of the DD, it gives an overview of the main components of the system and the relationship between them providing the most relevant views. This section also focus on the main architectural styles and patterns adopted in the design of the system.
- **Section 3: User interface design**: In this section are exposed again the UI mockups presented in the RASD document. In addition, UX diagrams are provided to better understand the paths that each user is able to follow while using the system.
- **Section 4: Requirements traceability**: This section associates the decision taken in the RASD with the ones taken in this DD.
- **Section 5: Implementation, integration and test plan**: Here is specified in which order the different components of the application are developed and integrated with each other. This section also specifies a strategy to follow to correctly test the implementation of the system.
- **Section 6: Effort spent**: This section includes information on the number of hours each group member worked for this document.

2. Architectural design

A. Overview

The Customers Line-up system has a four-tier architecture that can be grouped into 3 logical layers: presentation, application and data.

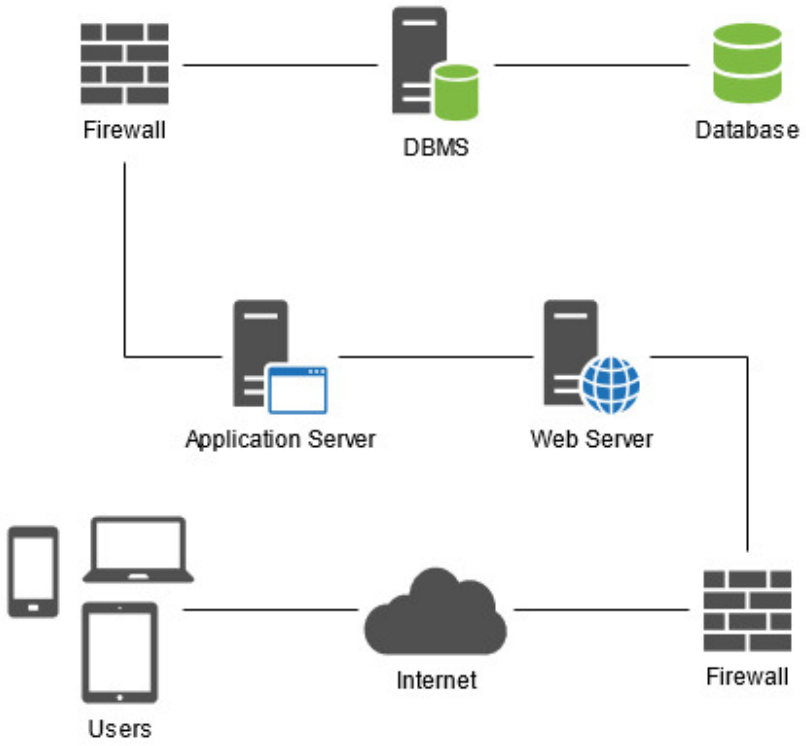


Figure 1: Physical architecture diagram

On the client side there is nothing but a browser used to connect to the web server and to dynamically update the *Web App View* when needed (only for minor changes).

The web server, on the other hand, acts as a middleware between the client interface and the system logic, communicating with the user via the standard HTTP protocol, receiving requests and providing adequate responses. To do this, this node also interacts with the application server to submit changes in user data and to request pieces of information needed to generate and update the UI.

Finally, the business logic of the system resides on the application server (*Back-end*). This node is able to manage the connection with the DBMS and takes care of carrying out the necessary processing for the correct functioning of the system.

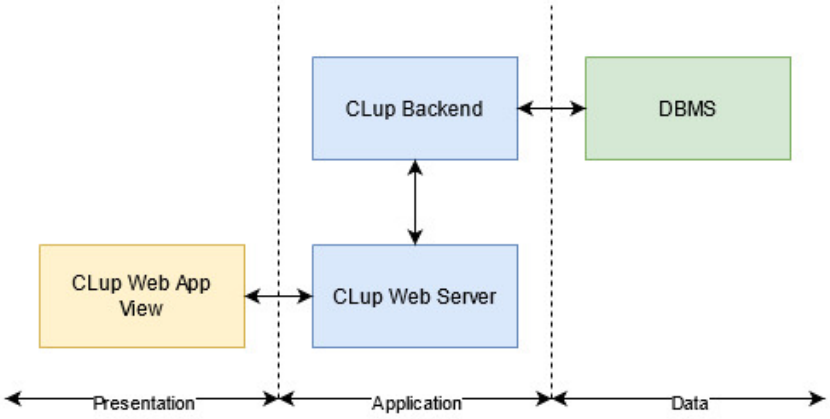


Figure 2: High level layers diagram

B. Component view

The following diagrams show the main components of the CLup system and the interfaces through which they interact.
For clarity, the first diagram shows a high level view of the components, which are then further explored individually.

The system exposes a RESTful API with multiple public endpoint and resources, some of them require a proper authentication and authorization to be used.

B.1. High level component

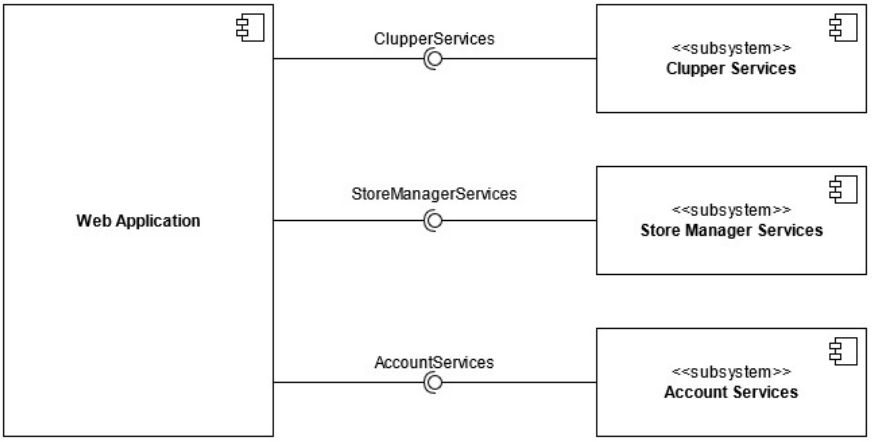


Figure 3: High level component diagram

The client side consists of a single component that refers to the web application, the only access point to this system.

The server side, on the other end, is made of three subsystems:

- **Clupper Services**: provides access to the basic services of the application after the login, this services are reserved to the cluppers.
- **Store Manager Services**: provides access to the managerial services of the application after the login, this services are reserved to the store managers.
- **Account Services**: provides support to registration, login and logout operations for any type of user.

B.2. Clupper Services projection

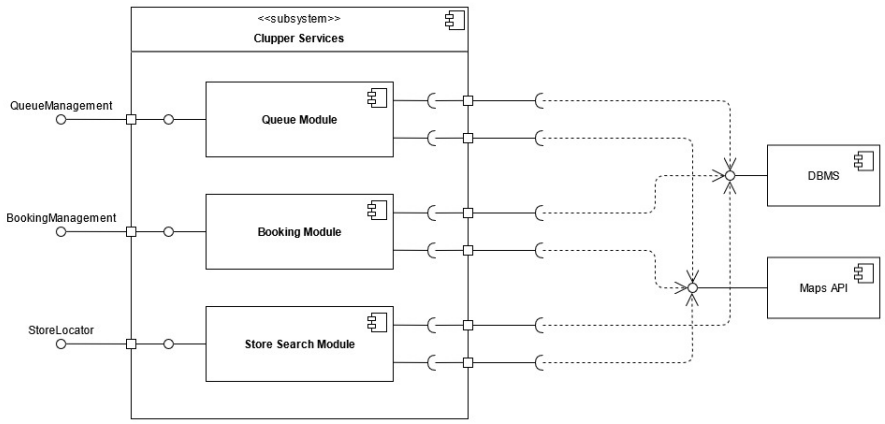


Figure 4: Clupper Services projection diagram

The Clupper Services subsystem contains three components:

- **Queue Module**: offers the *QueueManagement* interface to handle all the operations related to the join the queue function.
- **Booking Module**: offers the *BookingManagement* interface to handle all the operations related to the book a visit function.
- **Store Locator**: offers the *StoreLocator* interface that allows the client to retrieve information about the stores.

In order to fulfill their goals, these components need to communicate with the DBMS and the Maps API through the corresponding interfaces.

B.3. Store Manager Services projection

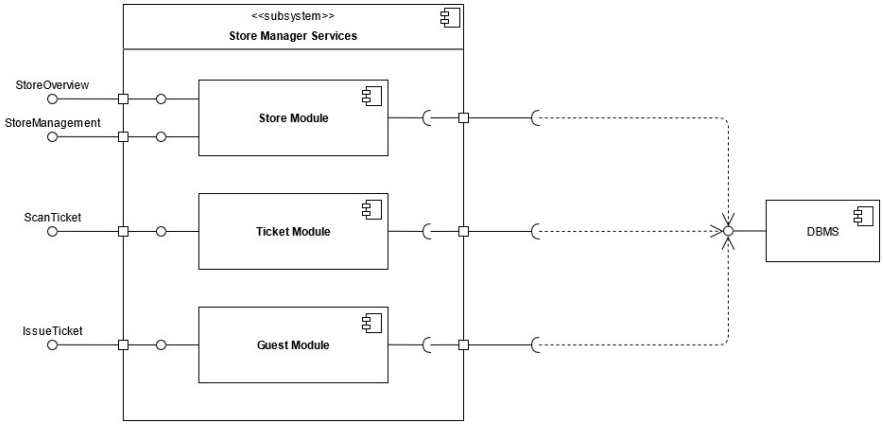


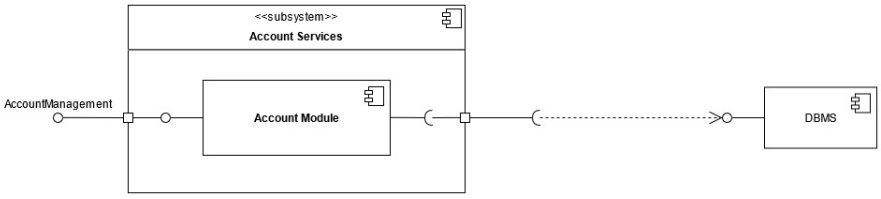
Figure 5: Store Manager Services projection diagram

The Store Manager Services subsystem contains three components:

- **Store Module**: offers the *StoreOverview* and the *StoreManagement* interfaces used to receive store status updates and to edit the store maximum capacity respectively.
- **Ticket Module**: offers the *ScanTicket* interface to receive feedback on scanned tickets.
- **Guest Module**: offers the *IssueTicket* interface to request or delete a queue ticket for a guest.

In order to fulfill their goals, these components need to communicate with the DBMS.

B.4. Account Services projection



The Account Services subsystem contains one component:

- **Account Module**: offers the *AccountManagement* interface to handle the operations of register, login and logout.

In order to fulfill its goals, this component needs to communicate with the DBMS.

B.5. ER diagram

The following diagram provides a graphical representation of the conceptual model of the database.

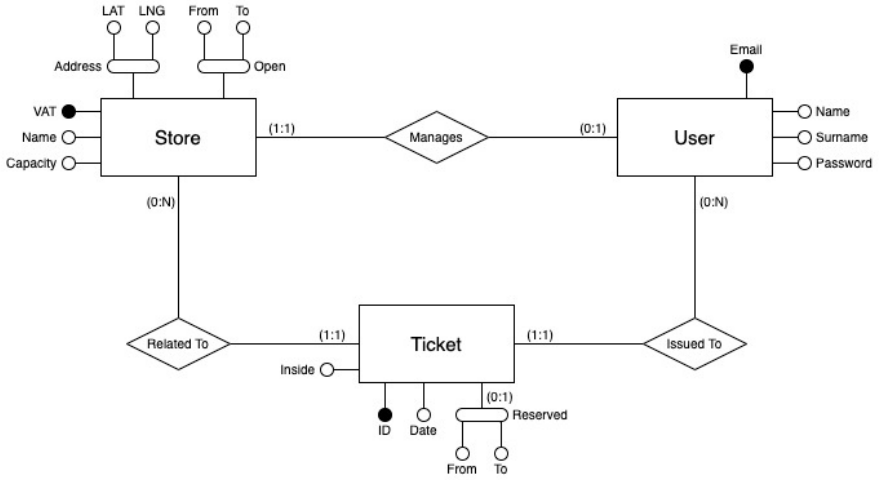


Figure 6: Entity-relationship diagram

C. Deployment view

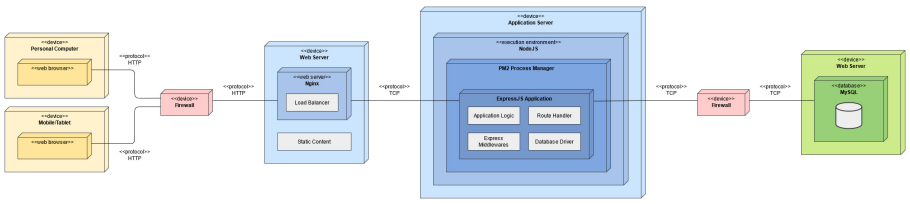


Figure 7: Deployment diagram

The system architecture is divided in 4 tiers:

- The first tier is the *client tier*: it is composed by any device capable of rendering a web page (smartphones, tablets, PC). It communicates with the web tier through the HTTP protocol.
- The *web tier* contains the web server implemented with Nginx platform. This tier is provided with a load balancer to distribute the load between the multiple instances of the application tier.
- The third tier is the *application tier*. The CLup backend is built on an Express application which is managed by a docker-like process manager to spawn multiple instances. The execution environment is the NodeJS runtime.
- The *data tier* is the fourth tier composed by the Database server.

C.1. Recommended implementation

- **Client tier**: The client web browser may be an arbitrary one,the only constraint is that it must be recent enough to render HTML5 and CSS3 web pages and execute Javascript.
- **Web tier**: The web tier must be implemented with Nginx web server. A load balancer is also needed to distribute the load between the multiple instances of the application. It is strongly recommended to setup the load balancer to use a Round Robin algorithm to distribute the load, to avoid overloading a single application instance.
- **Application tier**: The runtime engine where the backend lives is NodeJS and a docker-like process manager (PM2) is used to spawn multiple instances of the application, which uses ExpressJS to expose services.
- **Data tier**: The database is relational and is implemented using MySQL.

D. Runtime view

The following sequence diagrams describe the interactions between the main components of the product when utilizing the most common features.
This is still a high-level description of the actual interactions so that they can be slightly modified during the development process.

D.1. User login

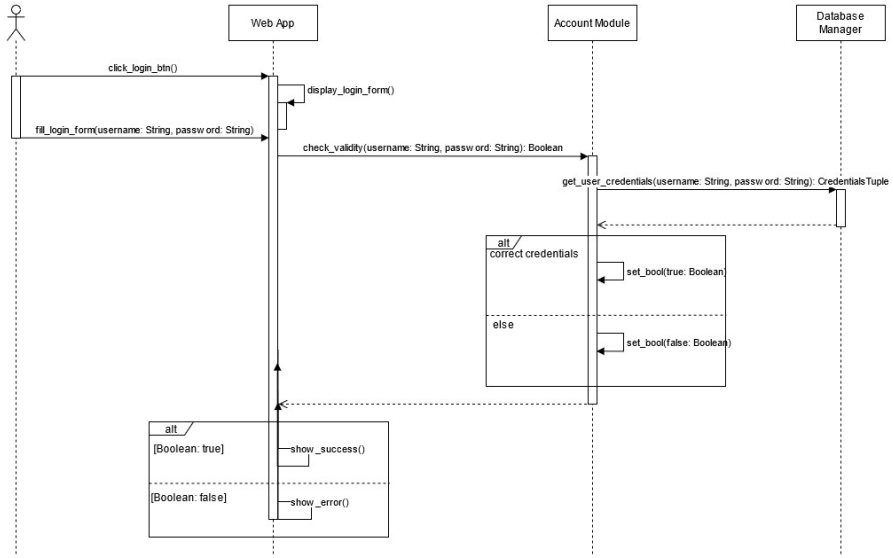


Figure 8: User login sequence diagram

D.2. User book a visit

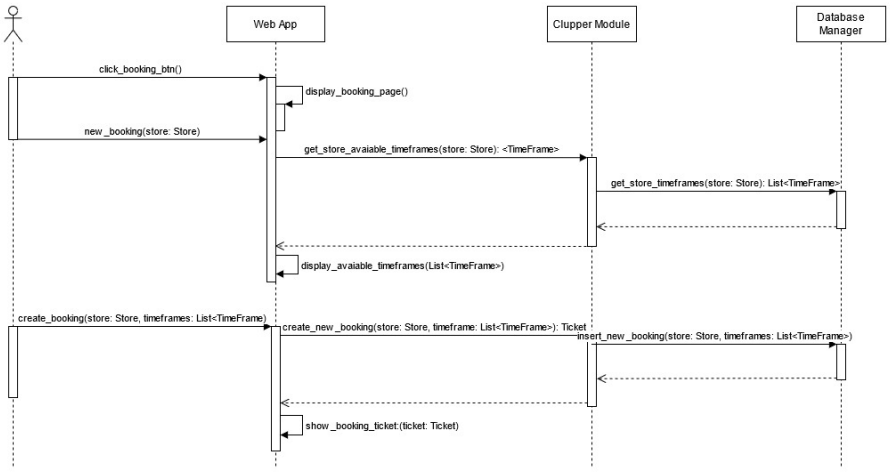


Figure 9: User book a visit sequence diagram

D.3. Store manager scan ticket

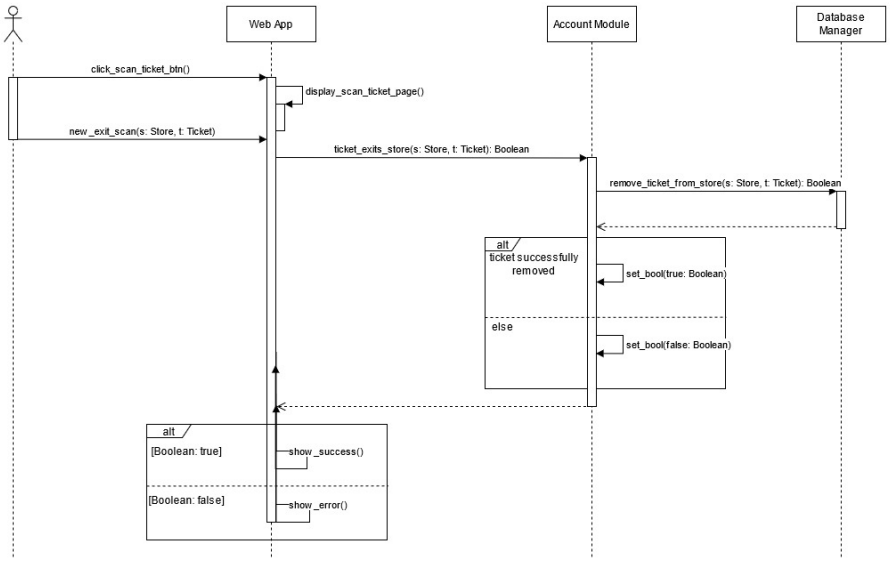


Figure 10: Store manager scan ticket sequence diagram

E. Component interface

These diagrams show the main methods provided by each component:

- The *ClupperServices* interface provides every method the clupper could need to access every functionality of the application. In particular there are methods to join the queue, book a visit and find stores.
- Like ClupperServices, *StoreManagerServices* provides all methods a store manager needs to manage the store. In fact, he's allowed to edit the store capacity, check the number of customers inside, in line and with a reservation and lend out physical tickets to guests.
- *AccountServices* is an interface accessible by both users and visitors. It let's the users login and logout and allows the registration of new cluppers or store managers.

E.1. ClupperServices interface

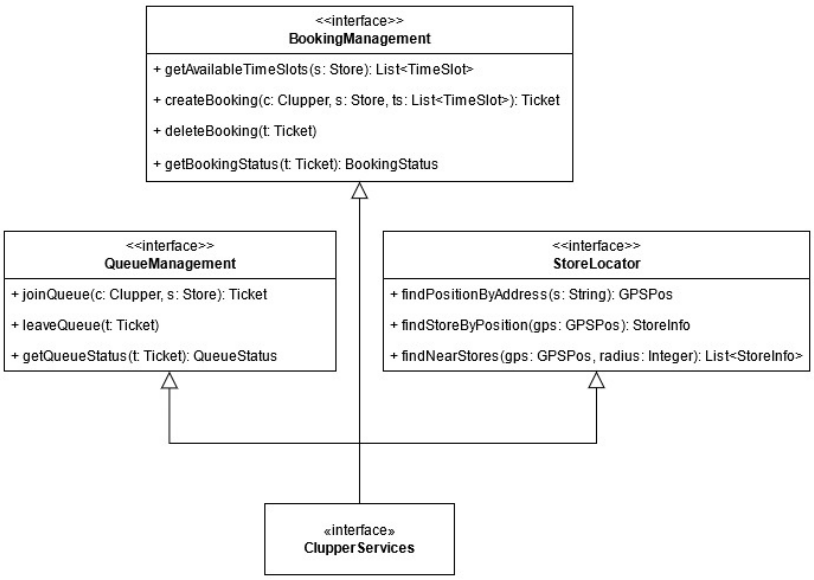


Figure 11: ClupperServices interface diagram

E.2. StoreManagerServices interface

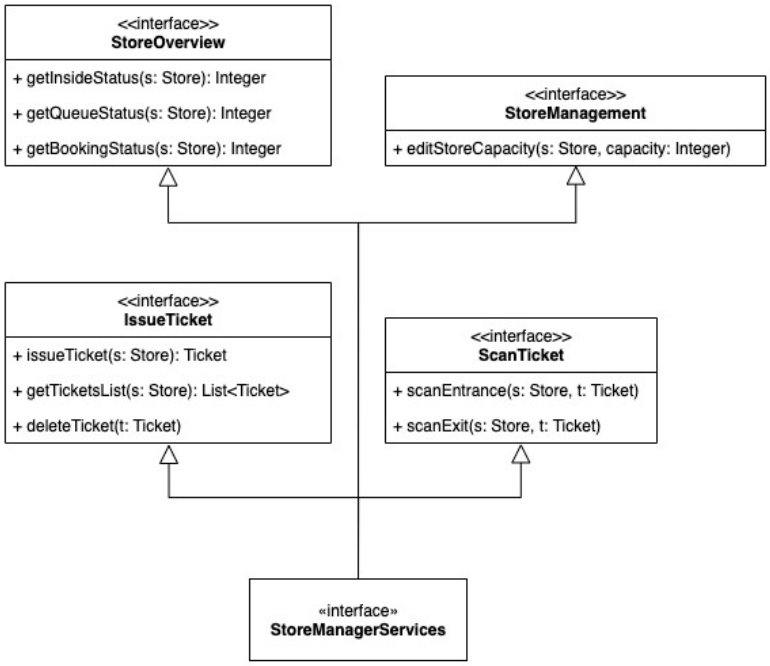


Figure 12: StoreManagerServices interface diagram

E.3. AccountServices interface

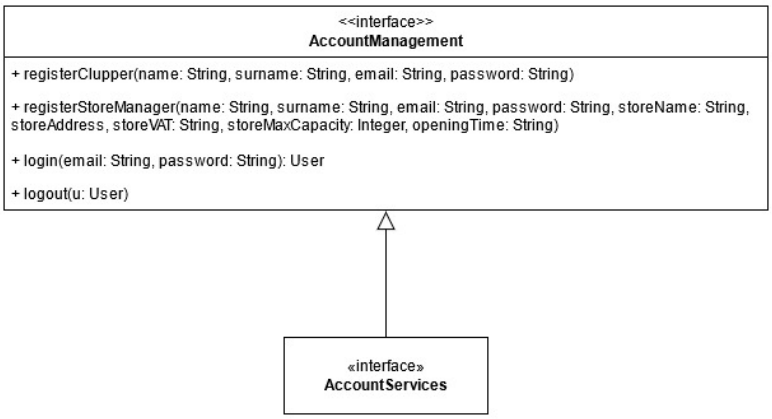


Figure 13: AccountServices interface diagram

F. Selected architectural styles and patterns

F.1. MVC pattern

The Model-View-Controller pattern is used in order to obtain a clear division between the internal representation of information, the way in which that information is presented to the user and the business logic of the system.
This is one of the most common and effective ways to drastically reduce the level of coupling between the various parts of the system.

F.2. RESTful API with JSON

The application layer of the system offers a set of services that each client is able to address via HTTP protocol.
As prescribed by the REST principles, for each incoming request the server provides a response that can also include a JSON representation of one or more objects.

F.3. Other design patterns

- **Facade pattern**: supplies a single interface to a set of interfaces within the system.
- **Factory pattern**: exposes a method for creating objects, allowing subclasses to control the actual creation process.
- **Observer pattern**: notifies the subscribed objects when the state of the observed object is updated.
- **Proxy pattern**: allows object level access control by acting as a pass through entity or a placeholder object.
- **Singleton pattern**: provides a single instance of a class that is able to coordinate all the actions across the system.

G. Other design decisions

G.1. Maps API

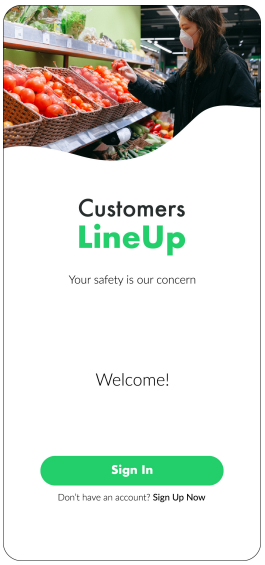
To implement some of the system functions (which require GPS information), the use of an external service is unavoidable.
When dealing with third party software it is important to be able to trust its source, for this reason CLup will use the API offered by Google Maps.
In particular, as this system will be developed as a web application, the *Google Maps Javascript API* is recommended.

3. User interface design

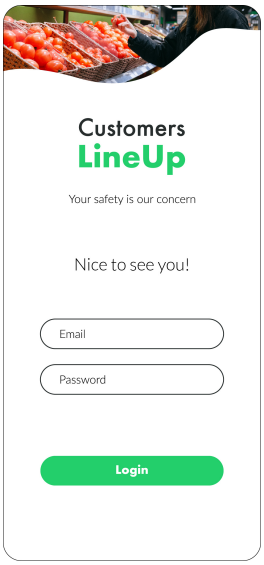
A. UI mockups

The following mockups (already presented in section *3.A.1. User interfaces* of the RASD) provide an idea of how the web application offered by CLup will look like.

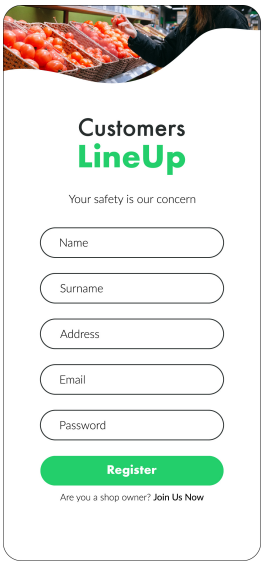
Since the main devices that will use the system are portable, a mobile-first approach is recommended in the UI development, however, the client view must be designed responsively to best fit any screen size.



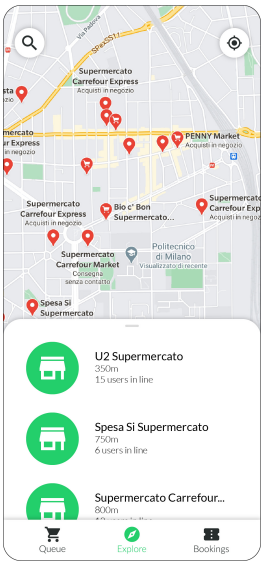
Home page



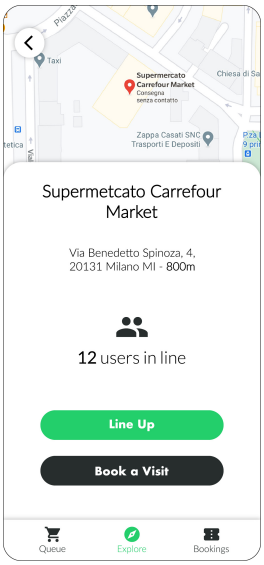
Sign in page



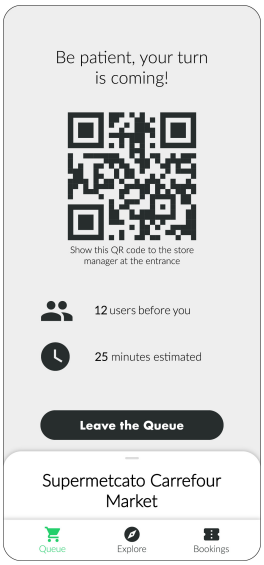
Sign up page



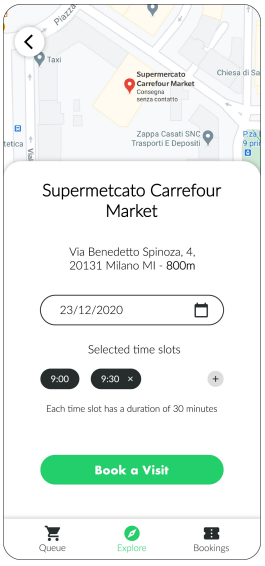
Map page



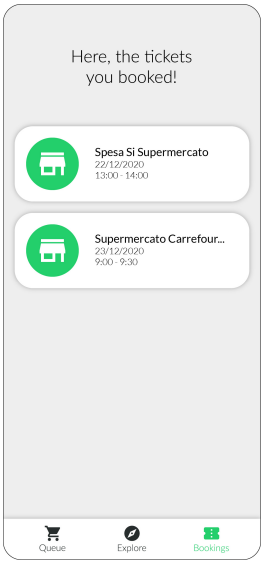
Store page



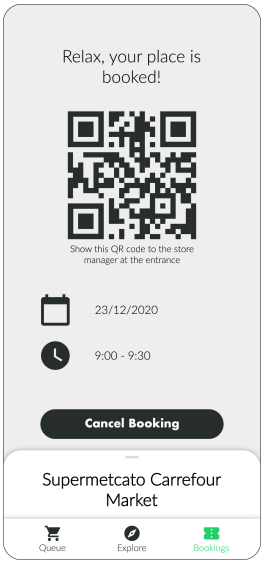
Queue page



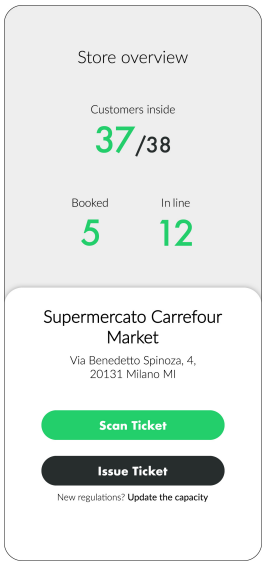
New reservation page



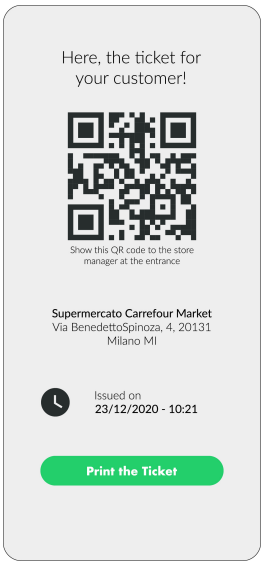
Bookings page



Reservation page



Store overview page



Issued ticket page

B. UX diagrams

The following UX diagrams provide additional information about the journey of the various users inside the CLup web application.

The paths represented in the following diagrams take for granted the absence of exceptions of any kind. Each exception is handled as explained in the section *3.B.1. Use cases* of the RASD.

Using the *back* button provided by the web browser or the device will result in the default behaviour (previous page in history) and will therefore be ignored in this context.

B.1. Visitor

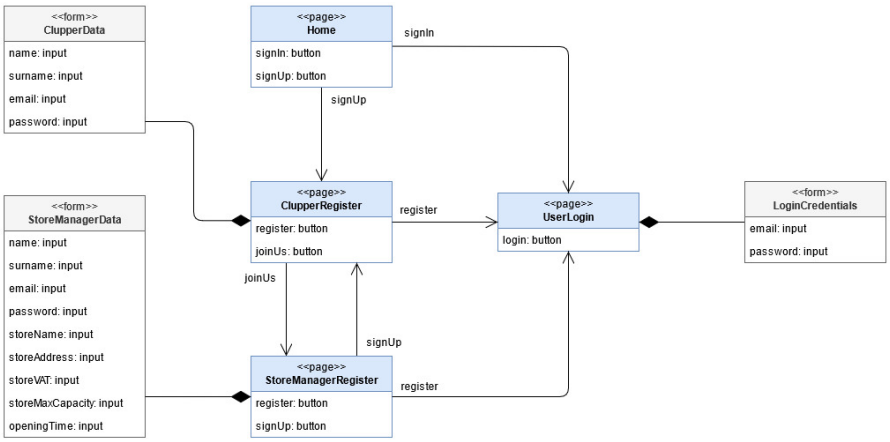


Figure 18: Visitor diagram

The direct path from the *Home* page to the *UserLogin* page is considered as the default behaviour and therefore omitted in the next diagrams. The *LoginCredentials* form is also not reported for clarity.

B.2. Clupper

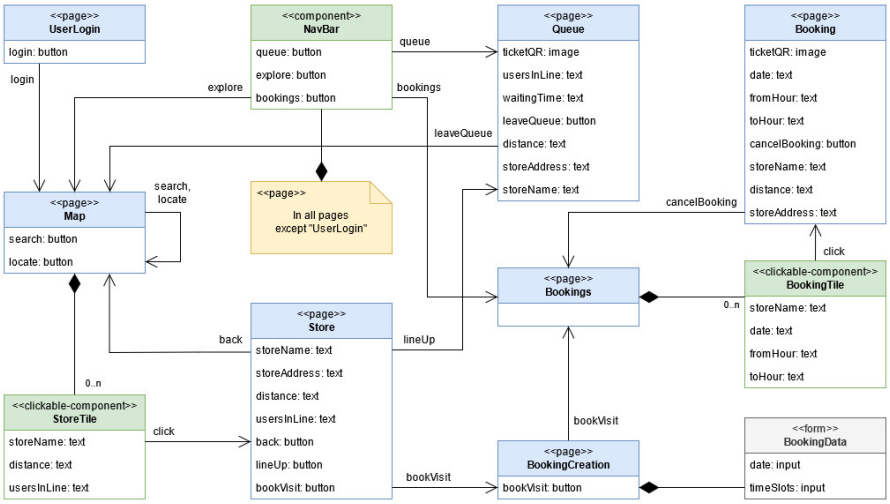


Figure 19: Clupper diagram

The *NavBar* component is included in every page accessible by the clupper except the "UserLogin" page, the arrow is omitted to improve readability.
The *StoreTile* and *BookingTile* can be included multiple times on the same page (with different data), they are classified as *clickable* since they behave like a button.

B.3. Store manager

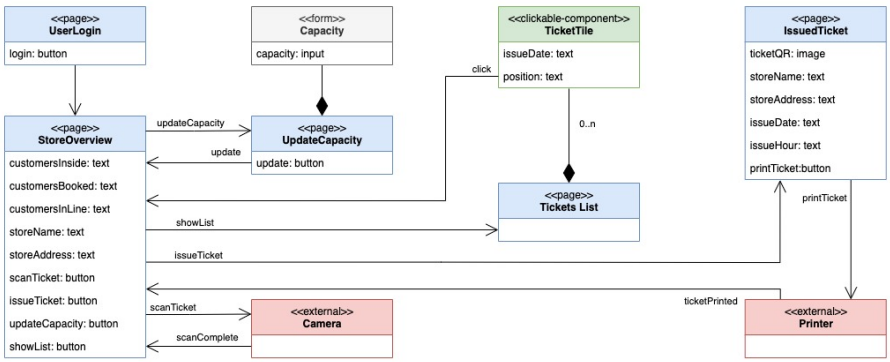


Figure 20: Store manager diagram

Camera and *Printer* represent the use of that specific external component (required for the correct functioning of the application) and have been included in the diagram to better clarify the sequence of actions that the store manager will perform.

4. Requirements traceability

While making the design choices presented in this document, the main objective was to fulfill in a complete and correct way the previously identified goals of the system.

Here is provided a mapping between the goals defined in the RASD and the system components illustrated in the DD.

- **G1**: Allow a clupper to join a store queue without having to physically approach the store.
  - Account Services (*Account Module*)
  - Clupper Services (*Queue Module*)
- **G2**: Allow a guest to join a store queue by requesting it to the store manager.
  - Account Services (*Account Module*)
  - Store Manager Services (*Guest Module*)
- **G3**: Allow a clupper to leave a store queue before entering.
  - Account Services (*Account Module*)
  - Clupper Services (*Queue Module*)
- **G4**: Allow a clupper to book a visit to the store.
  - Account Services (*Account Module*)
  - Clupper Services (*Booking Module*)
- **G5**: Allow a clupper to cancel a reservation to a store before entering.
  - Account Services (*Account Module*)
  - Clupper Services (*Booking Module*)

- **G6**: Allow a clupper to approach at the right time the store he has a ticket for.
  - Account Services (*Account Module*)
  - Clupper Services (*Queue Module*, *Booking Module*)
- **G7**: Allow a store manager to have an overview of the store.
  - Account Services (*Account Module*)
  - Store Manager Services (*Store Module*)
- **G8**: Allow a store manager to regulate the entrances to the store.
  - Account Services (*Account Module*)
  - Store Manager Services (*Store Module*, *Ticket Module*)
- **G9**: Allow a store manager to regulate the exits from the store.
  - Account Services (*Account Module*)
  - Store Manager Services (*Store Module*, *Ticket Module*)
- **G10**: Allow a guest to leave a store queue before entering.
  - Account Services (*Account Module*)
  - Store Manager Services (*Guest Module*)

5. Implementation, integration and test plan

5.A. Implementation plan

This section defines the implementation order for the different components of Customers Line-up that must be followed by the development team.

For the implementation of this application a *bottom-up* approach will be used. This approach has the purpose of developing components that can be used in different situations and facilitates bug tracking by allowing incremental testing.

The first element that must be implemented is *Database Connection*, the internal component that manages the interaction between the system and the *DBMS* and provides a set of functionalities used by other components.

Then, it is possible to develop the components that provide the services offered to users by the system.
These elements are fundamentally independent of each other, so a deviation from the order in which they are presented in this document is permitted.

Starting from *Clupper Services*, its subcomponents are implemented following the order in which each user will interact with them, thus defining an indirect information dependency:

1. Store Search Module
2. Queue Module
3. Booking Module

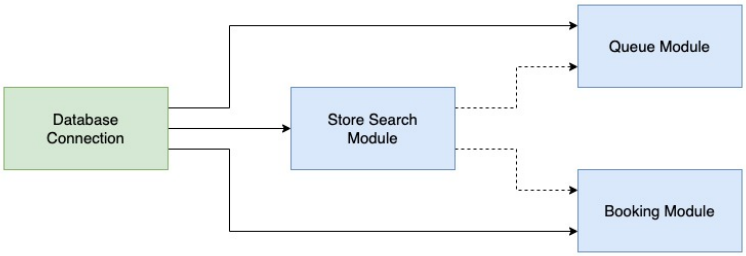


Figure 21: Clupper services diagram

Like before, inside *Store Manager Services*, the subcomponents implementation is done following the dependency induced by the user interaction with the system:

1. Store Module
2. Ticket Module
3. Guest Module

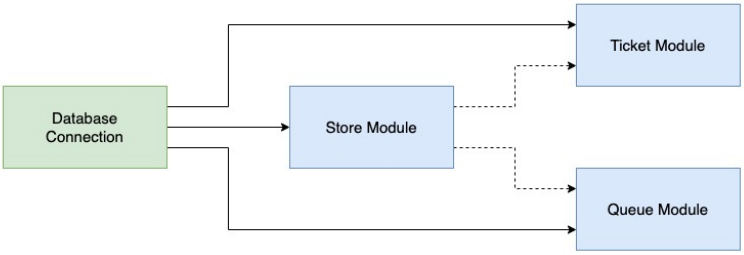


Figure 22: Store manager services diagram

Finally, the *Account Services* component, can be implemented as a single block as it provides standard functionalities used for authentication and authorization purposes.

Figure 23: Account services diagram

The implementation of the user interface can be performed in parallel with the implementation of the previous components after defining a set of offered APIs.

5.B. Integration and test plan

The implementation of the application must be accompanied by a meticulous testing phase.

In addition to unit tests, the development of a component requires a series of integration tests with the part of the system already implemented.
For this reason the order of integration and testing strictly depends on the order of implementation.

As said before, the style adopted for this system allows the development of the view components in parallel.
This advantage can be exploited to perform tests of each functionality without having to wait for the whole system to be completed.

After the complete integration, the application must be tested as a whole to verify that the requirements are actually satisfied. This is done by means of system testing.

6. Effort spent

Pair programming

Topic	Hours
Architectural design diagrams	3.5h

Ferrara Alessandro

Topic	Hours
Component Interface Diagrams	1.0h
Sequence Diagrams	1.0h
Section 2	2.0h
Section 5	2.5h

Fratus Lorenzo

Topic	Hours
UX diagrams	2.0h
DD structure	1.0h
Sections 2 and 3	2.5h
Section 1	2.0h
Section 4	1.0h

7. References

- Software Engineering II course slides
- Decreto Ministeriale