POLITECNICO
MILANO 1863

Software Engineering II project

CLup

Design Document

Authors:

Alberto MAGGIOLI
Lorenzo PORETTI

Professor:

Matteo Giovanni Rossi

Version 2.0

Contents

1. Introduction

Because of the pandemic situation due to Covid-19 there is a need to make sure that people can be spaced out and ensure their safety. For this reason, supermarkets also need to ensure social distancing.

1.A Purpose

This document aims to describe design and architecture of CLup system relying on the previous Requirement Analysis and Specification Document.

In this document have been further investigated architecture choices focusing on the motivations that led to them.

In the document are described in a detailed way all the components and interfaces of the system, how those components interact with each other and how the GUI must look like.

This document is of great importance for the development of the whole system and the developers will have to follow it.

1.B Scope

The aim of this project is to design an application which help store managers to regulate the influx of people in the building monitoring entrances to comply with the coronavirus safety regulations.

Moreover the application has the goal to prevent people from waiting outside the store safeguarding clients health and not wasting time.

The application has several features:

- The user can take a ticket with which he is placed in a virtual queue. Right before his turn he is notified by the application so that he can go to the store on time.
- The user can book a visit to the supermarket at the time and date he prefers, according to the remaining availabilities.
- The application suggests to the user in the booking process some different times with a smaller number of people inside the store, other supermarkets of the same chain or other chains.
- The user in the booking process can also insert in detail the products he/she intends to buy so that the application can control and distribute in the best way the users inside the store.
- The application is able to predict the duration of the visit by inferring on the duration of a particular customer's past visits. Alternatively, the duration of the visit can be specified by the user during booking.

The application should be very simple to use, as the range of users include all demographics and it should be real time application to let user to allow clients to be informed.

1.C Definitions, Acronyms, Abbreviations

1.C.1 Definitions

Visit	A customer goes to the supermarket
Reservation	an arrangement that you make to have a visit in a supermarket.
Slot	Period of time in which the calendar is divided
QR code	It is a type of matrix barcode. A barcode is a machine-readable optical label that contains information about the item to which it is attached.
QR scanner	An object in charge to read QR codes

1.C.2 Acronyms

DD	Design Document
RASD	Requirement Analysis and Specification Document
QR	Quick response
DBMS	Database Management System
API	Application Programming Interface
GPS	Global Positioning System
OS	Operating System

1.C.3 Abbreviations

Rn	Requirement number n

1.D Revision history

Version	Date	Modifications
Version 1.0	05/01/2021	First Version
Version 2.0	17/01/2021	Added an introduction to the document Added some information regarding the encryption in security paragraph (Overview section) Change name of Web Server in Supermarket Server Change name of Supermarket in Supermarket App

1.E Reference Documents

- Specification Document: "R&DD Assignment A.Y. 2020-2021"
- Slides of the lectures
- Book: "Software engineering principles and practice", Hans Van Vliet
- "Requirement Analysis and Specifications Document CLup", Alberto Maggioli, Lorenzo Poretti

1.F Document Structure

The document is divided in seven sections:

1. Introduction: This section aims to describe the scope and the purpose of this document and to explain acronyms and abbreviations used in the document.

2. Architectural design: This part is one of the most important of the design document because in this section most of the design choices are described. The section begins with high level representation of the architecture and its tiers, then we describe the components of the system, starting in a high level way and then going in details of each subcomponent and interface.
   Some important interactions between components are then pointed out with sequences diagrams.
   The entire section terminates with a description of architectural styles and patterns and some other design decisions.

3. User Interface Design: this section presents mockups of how the application looks like and how the user navigates the application to obtain the functionality of the CLup system.

4. Requirements Traceability: It is described how the components described in the second section map the requirements listed in the RASD.

5. Implementation, Integration and Test Plan: In this section is explained in which order the components are developed during implementation phase and how test them.

6. Effort spent: this section is used to keep track of the hours spent by each team member to complete each section.

7. References: This section Includes all the references used to define the document

2. Architectural Design

2.A Overview

The system is based on a client-server architecture. The architecture organizes the system into three logical and physical computing tiers.

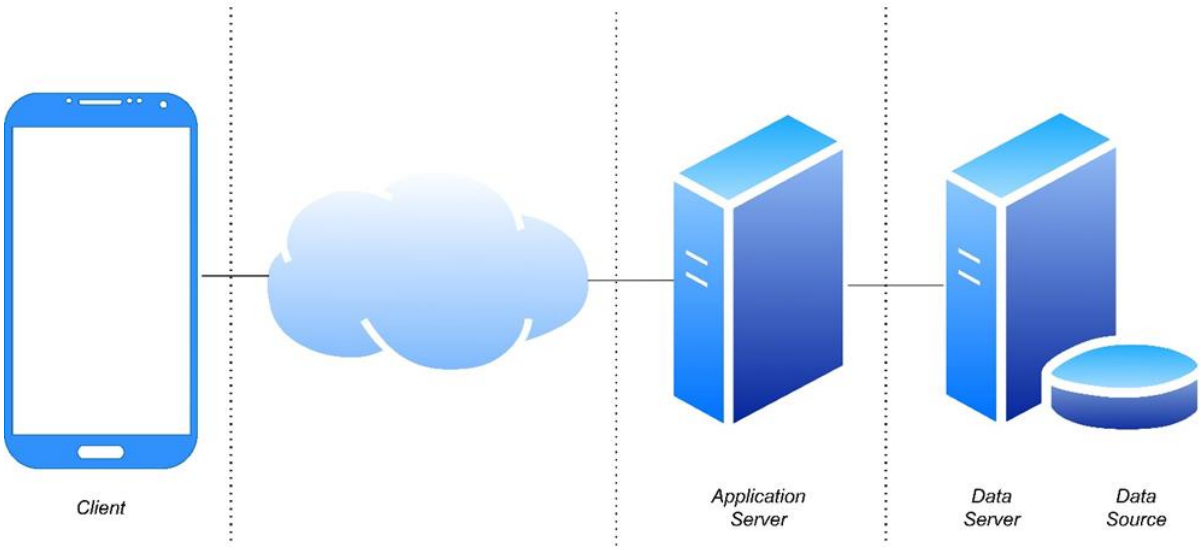
The first tier is in charge to interface with client, its main purpose is to display information and collect it from the user.

The middle tier manages the application level where information is collected and processed.

The last tier handles database management, where the information processed by the application is stored and managed.

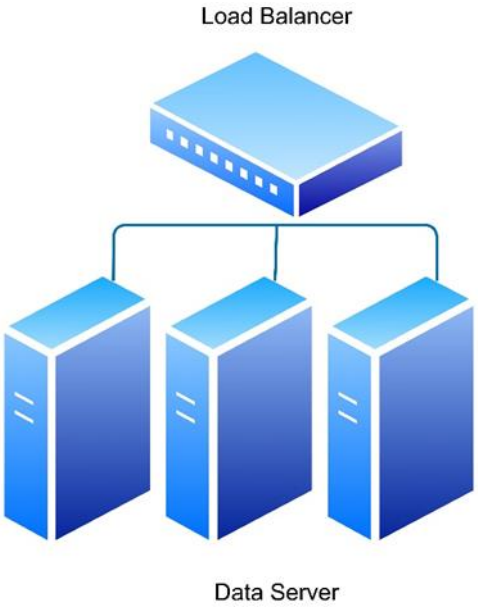
The decision taken has multiple benefits:

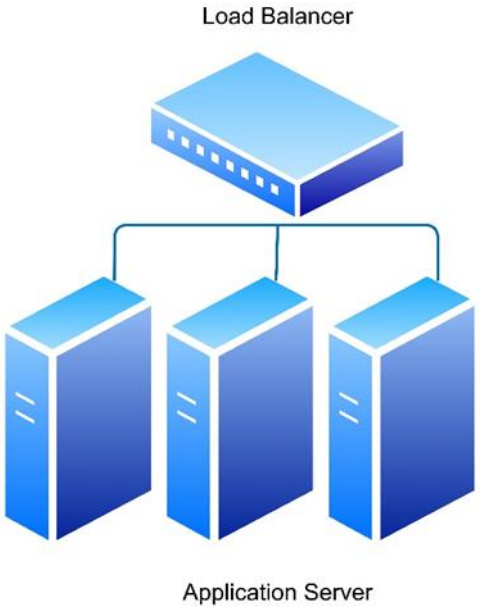
- Scalability: The architecture division into different parts made the system stabler while adapting to changes and upgrades.
- Security: Data access is possible only from the application layer. The application layer can be seen as a sort of internal firewall, preventing malicious exploits. Data are encrypted at rest and also they are encrypted in-Flight while data are being transmitted to provide confidentiality.
- Reliability: The independence between tiers guarantees that an outage of a component doesn't affect the entire system.
- Faster development: The development can be divided between different teams that can work in a parallel way.

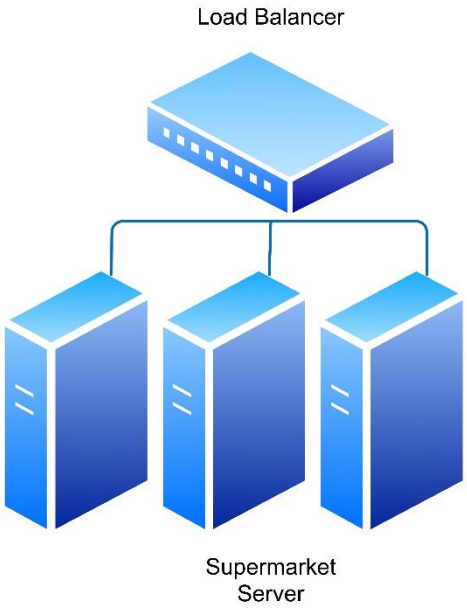


Three tier architecture

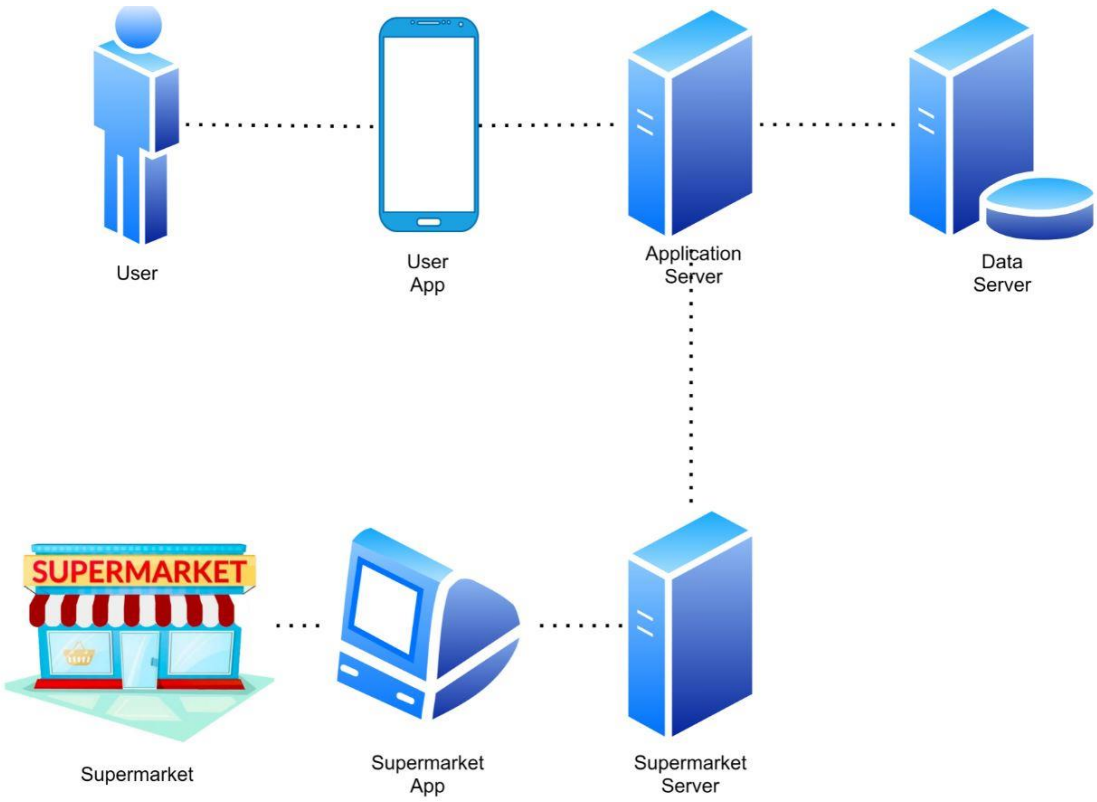
The configuration adopted is thin client which allows the user to have a simple logic application which doesn't affect performance of less powerful hardware. On the other hand all the logic application is processed by the application layer, this affects the application layer being busier. This choice requires also more communications between the client and the application server, these can be reduced using a cache memory between the client and the application server.







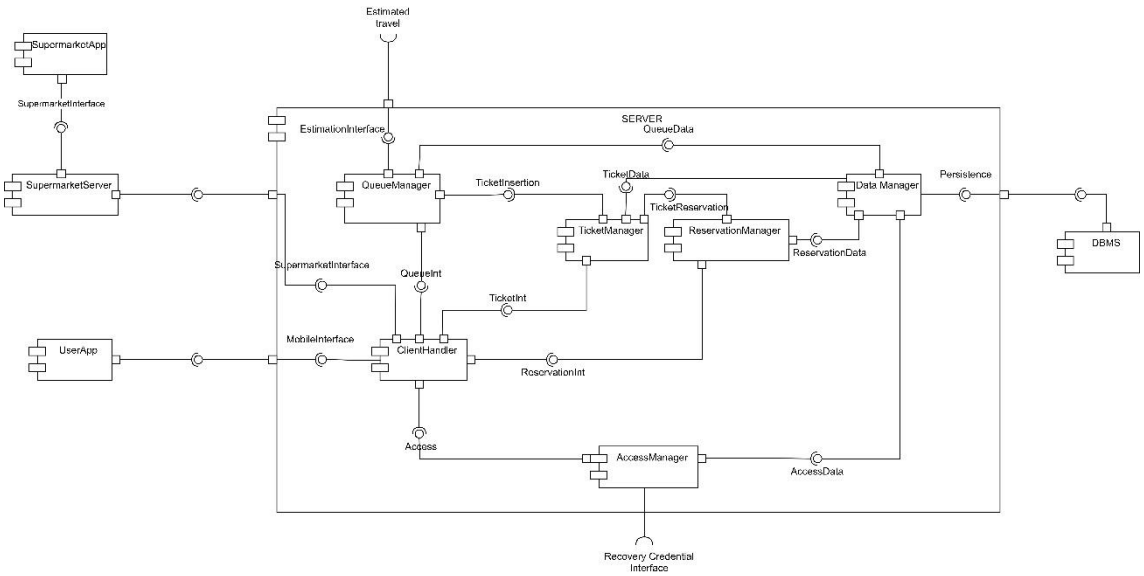
For both the Application tier and Data server, a load balancer parallelizes the application between different machines. This choice guarantees better performance thanks to less load for each server. A problem of a specific server doesn't affect the entire system. Maintenance is also better because the problem of a single server can be repaired while the system continues to work.



2.B Component View

The component diagram represents a high level view of the components of the system. The component diagram points out how the components interact with each other and it highlights what are the interfaces which let the interaction between components possible.

2.B.1 High Level Component



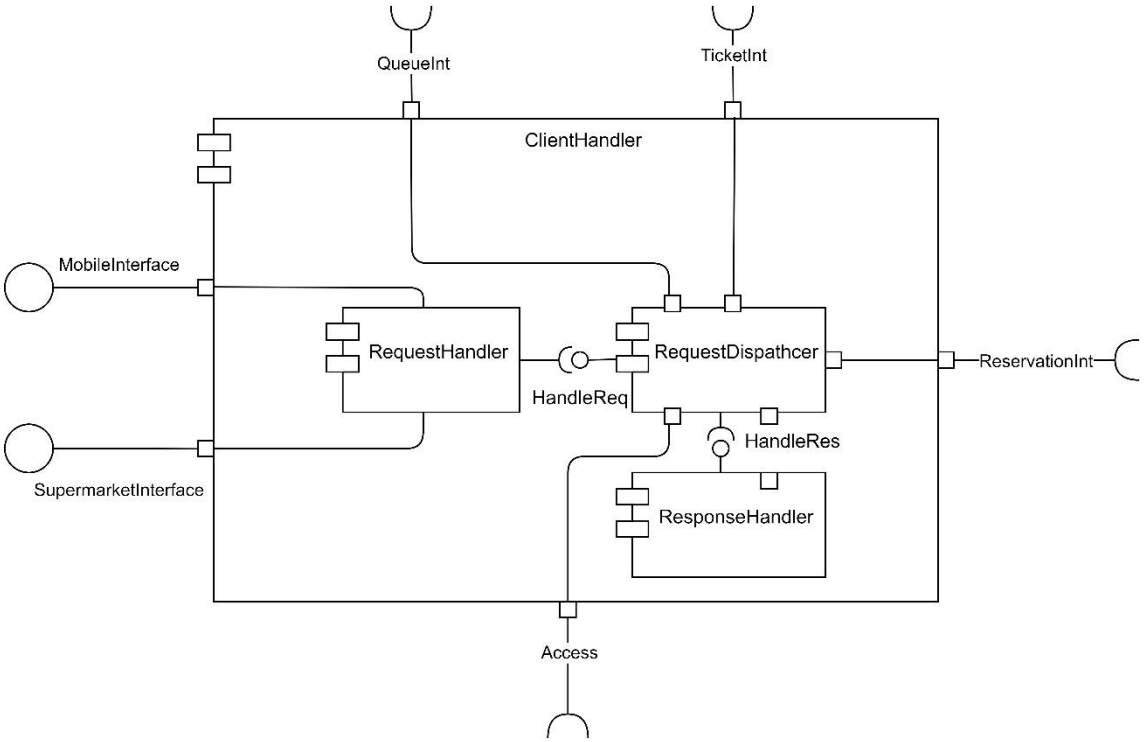
In this diagram there is a view of all the main components of the system. Each one is then explicated better pointing out also all the sub-components.

The main components of the system are:

- **UserApp**: it is the component that represents the application running on user's device
- **SupermarketApp**: It is the application that runs on supermarket devices.
- **SupermarketServer**: it is the component that interacts directly with the devices of the supermarket and communicates with the CLup system.
- **ClientHandler**: it is the component that handles requests coming from both the user and the supermarket. Our idea is to model a single client handler for both clients, in this way the component can manage the request independently from the source. All the requests are then shared to specific components which take charge of them.
- **QueueManager**: It is the component which deals with queue management. The main functionality is to keep track not only of the users actually in the queue but also of those who are inside the supermarket. The queue Manager is also in charge to notify users when it is their turn. This decision is taken thanks to an external API which calculate the time needed to reach the supermarket.
- **TicketManager**: It is the component which deals with ticket management. The main functionality is to handle ticket requests as for example a take a new ticket request or delete ticket request.

- **ReservationManager**: It is the component which deals with reservation management. The main functionality is to handle reservation requests as for example get free slots for a specific supermarket or book a reservation.
- **AccessManager**: it is the component in charge to handle access request of the user. The main management functionalities are Login management, registration management and credentials recovery management.
- **DataManager**: it is the component in charge to prepare information and execute the query to store and load information.
- **DBMS** : Database Management System (DBMS) is used to store and retrieve data directly from the database

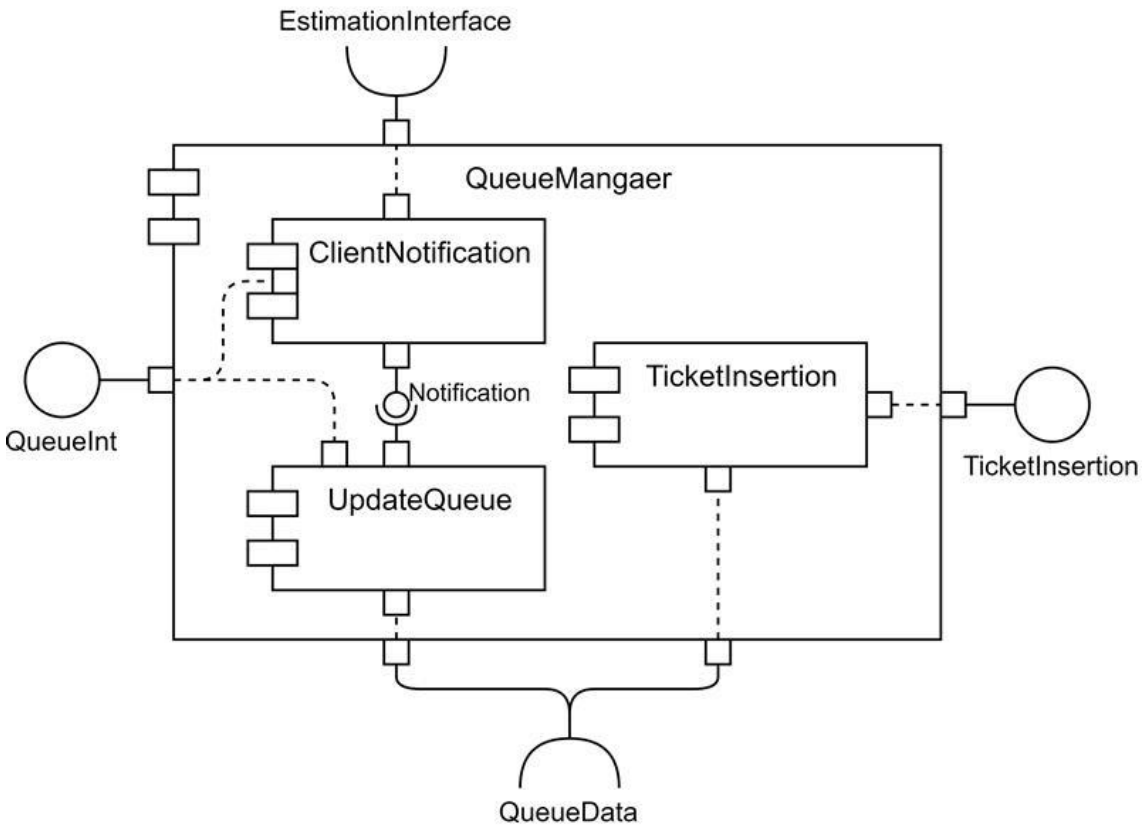
2.B.2 ClientHandler Component



The component consists in three sub-components:

- **RequestHandler**: This subcomponent handles the requests coming from both the supermarket server and mobile application.
- **ResponseHandler**: This subcomponent processes the system response to give a it to the client.
- **RequestDispatcher**: This subcomponent has a foundamental role in the application. It is in charge to dispatch the request to a specific component able to manage it.

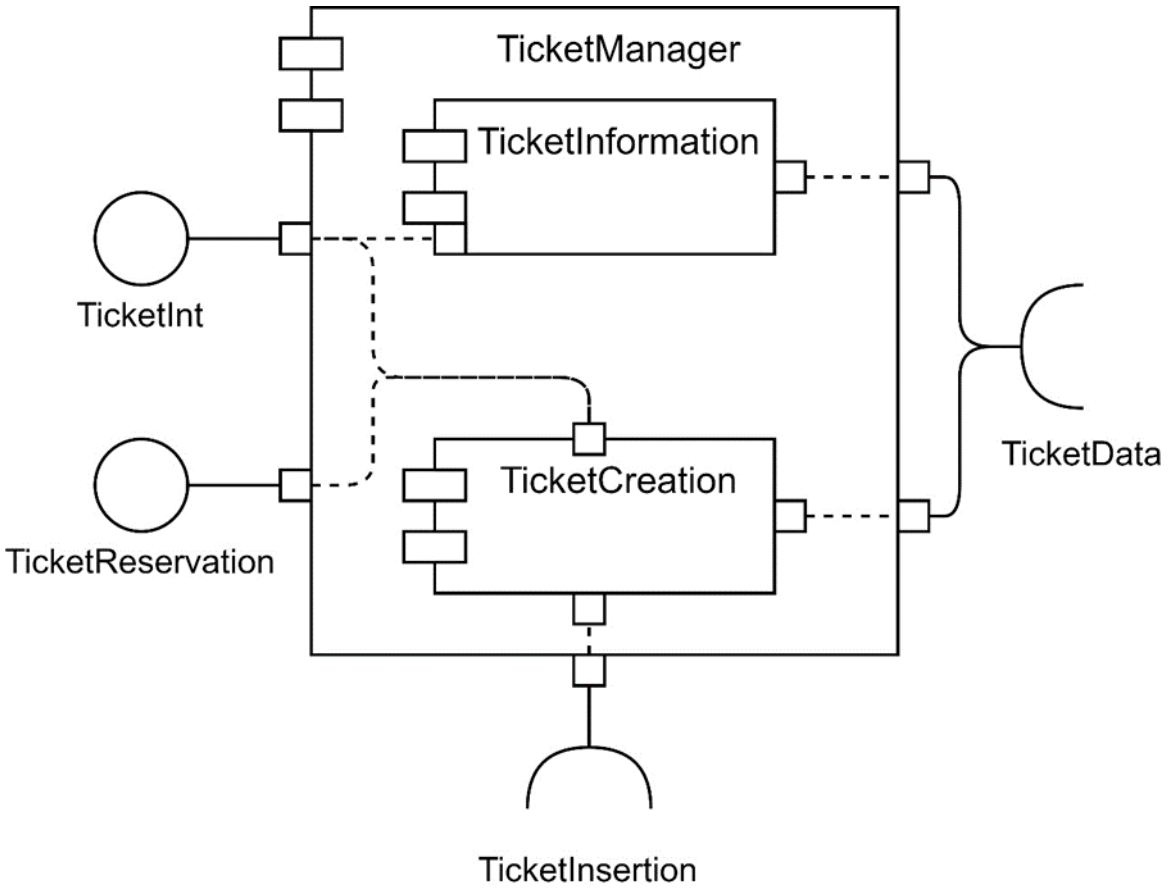
2.B.3 QueueManager Component



This component is composed of 3 sub-components.

- **ClientNotification**: This component has the function of managing the notifications to be sent to the user. Every time the queue is updated, this component takes care of checking the tickets for which a notification should be sent. In particular, for the first notification, the time remaining in the customer's turn is compared with the time taken by the user to reach the supermarket. The second notification is sent as soon as it is the user's turn. Note how the time needed to reach the supermarket is calculated thanks to an external API.
- **UpdateQueue**: This sub-component updates the queue whenever it receives information about a customer entering/leaving the supermarket. When the queue is updated, the component checks the notifications to be sent through the internal interface with the clientNotification sub-component.
- **TicketInsertion**: This sub-component has the function of inserting new tickets into the queue. The insertion of tickets into the queue is done with a FIFO policy, except for the tickets generated by reservations. In fact, these tickets are generated exactly at the time corresponding to the reservation and are therefore immediately called.

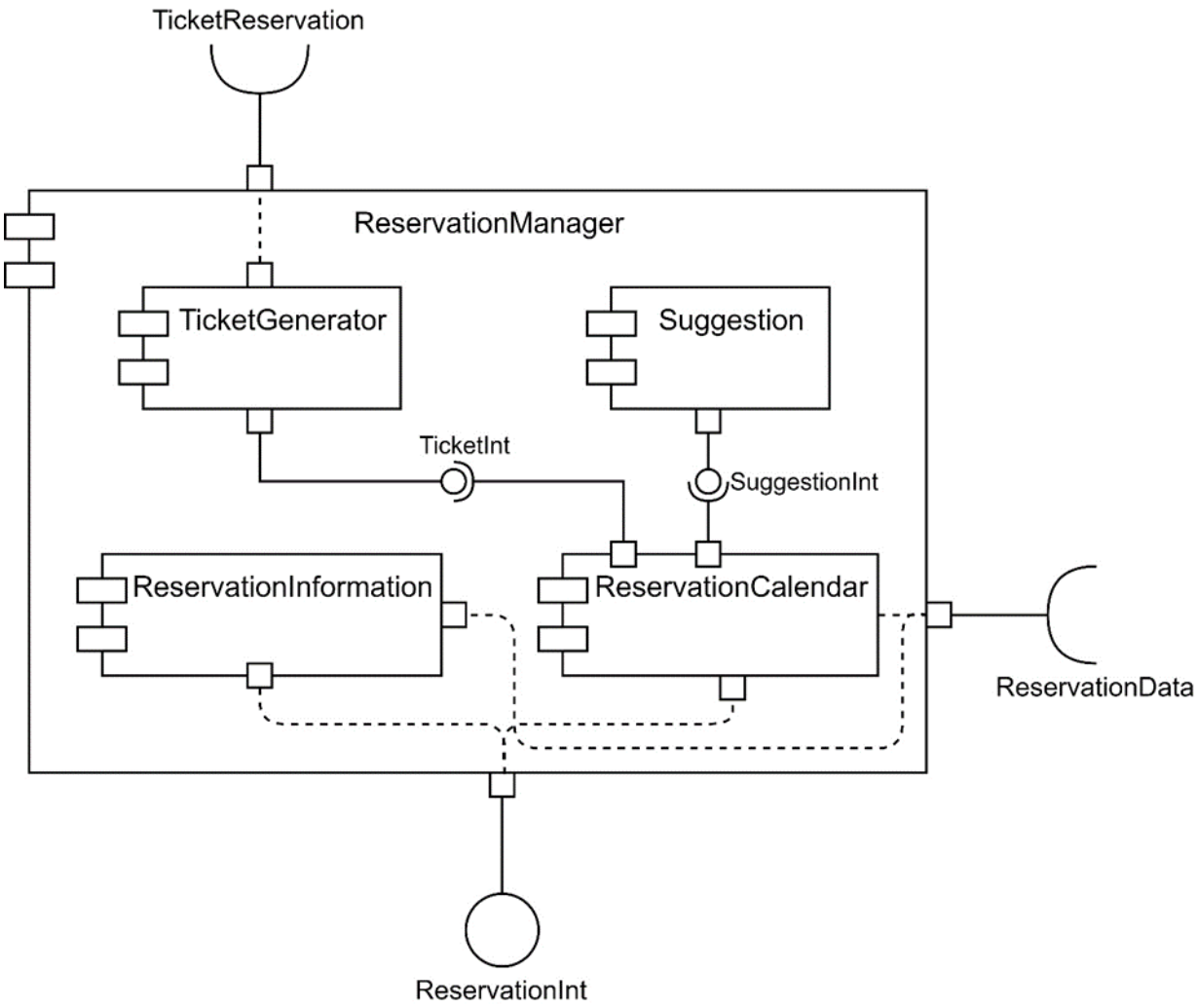
2.B.4 TicketManager Component



The component consists in two sub-components:

- **TicketInformation**: this sub-module allows the user to check information about his tickets, it also handles ticket elimination requests.
- **TicketInsertion**: this submodule allows the user to generate a new ticket, it is connected to the queue manager component through the TicketInsertion interface

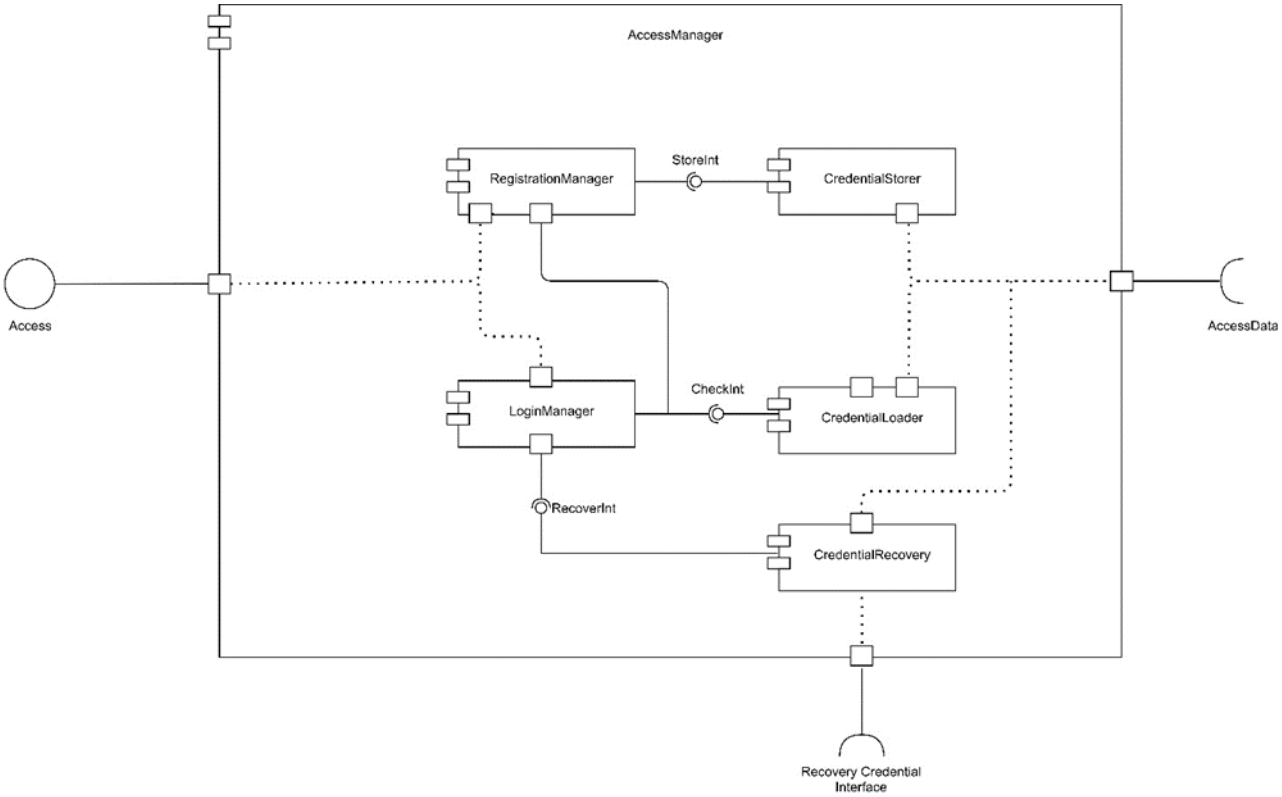
2.B.5 ReservationManager Component



The component consists in four sub-components:

- **TicketGenerator**: The task of this component is to create the reservation ticket as soon as the time comes. In particular the ticket is generated when it is the turn of the reservation. At that moment, through the TicketReservation interface, the ticket is created and added to the queue.
- **ReservationInformation**: This sub-component provides all the information to the user about his reservations. It also has the function of managing the reservation deletion requests. In order to access the reservation data the ReservationData interface is used.
- **Suggestion**: this component is responsible to identify booking suggestions to the client. Once the user has indicated his preferences, based on these preferences, the component searches for alternative slots so that the number of people inside the supermarkets can be optimally balanced.
- **ReservationCalendar**: this sub-component is responsible for managing the process of the "Book a visit" functionality. In particular, it shows to the user all the booking availabilities and it manages the preferences indicated by the customer during the booking process. Once the reservation is concluded, it saves it through the ReservationData interface. During the reservation, this component controls eventual suggestions through the SuggestionInt interface.

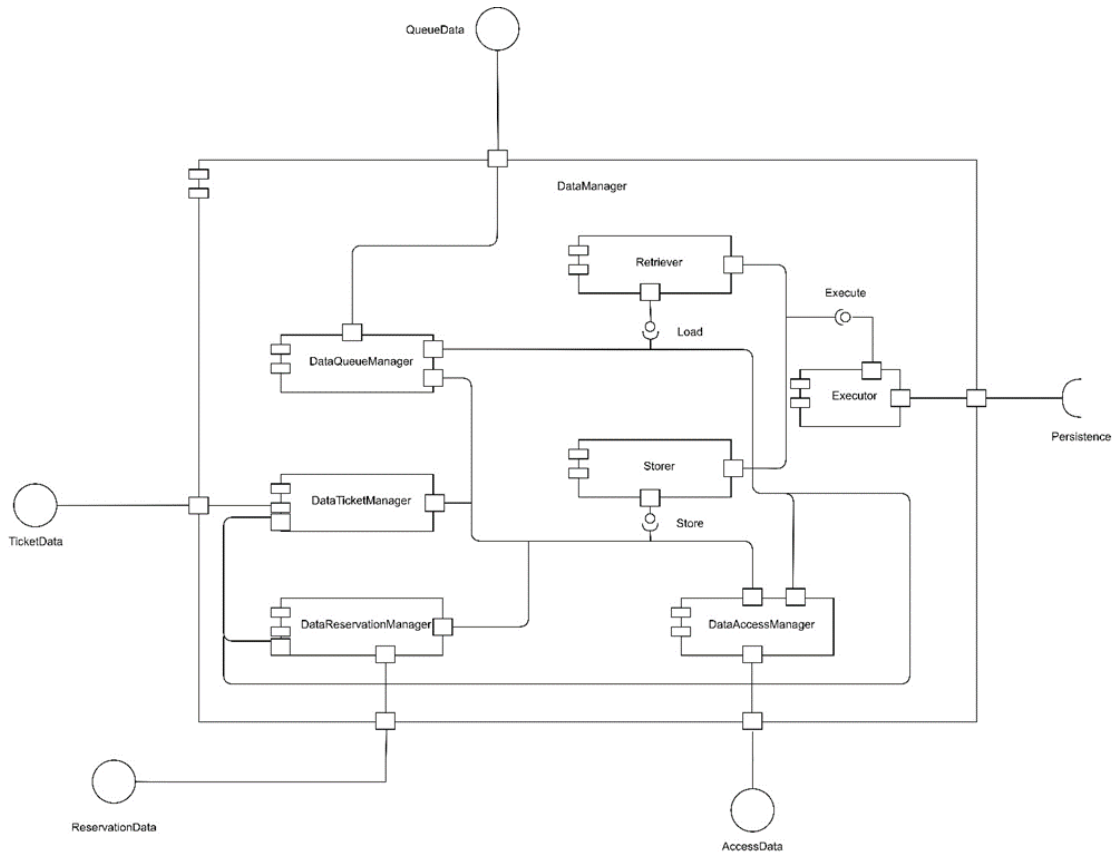
2.B.6 AccessManager Component



The component consists in five sub-components:

- **RegistrationManager**: it is the sub-component which has the task to manage the registration process. It processes user information given during registration. Once the registration information are processed, using the CheckInt the sub-component checks if credentials are not already used.
  The process ends storing user's credentials using the interface StoreInt.
- **LoginManager**: it is the sub-component in charge to handle the Login process.
  The main functionality is to process information given by the user and check the identity using the CheckInt. If the user is asking for credentials recovery, the sub-component is able to handle the request using the RecoverInt.
- **CredentialStorer**: it is the sub-component involved in the registration process. The main functionality is to store registration credentials.
  In order to store information in the database, the subcomponent uses AccessData interface.
- **CredentialLoader**: it is the sub-component involved in the registration and login processes. The main functionality is to load user information.
  In order to load information in the database, the subcomponent uses AccessData interface.
- **CredentialRecovery**: it is the sub-component that deals with the recovery of the credentials when the user needs to recovery the password of his account.
  This sub-component is directly interfaced with an external API which handle credentials recovery sending an email to the user. The external API is involved in the creation of a new password, once created, the CredentialRecovery sub-component manages to store the new password in the database using AccessData interface.

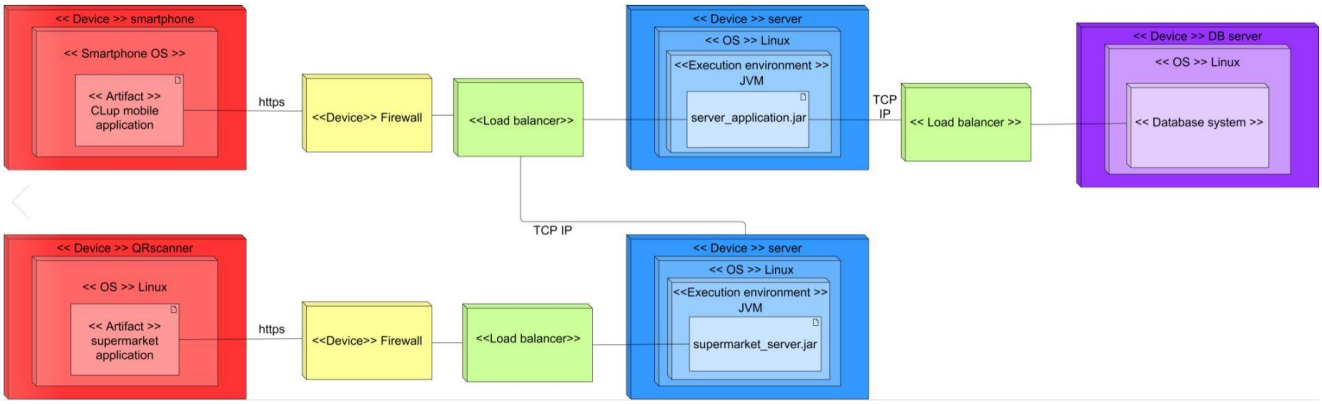
2.B.7 DataManager Component



The component consists in seven sub-components:

- **DataTicketManager**: it is the sub-component which provides methods to manage ticket data (create ticket, delete ticket, get ticket information…). It has two interfaces storer and retriever which respectively processes data to store and data to load.
- **DataReservationManager**: it is the sub-component which provides methods to manage reservation data (create reservation, get average visit time …). It has two interfaces storer and retriever which respectively processes data to store and data to load.
- **DataQueueManager**: it is the sub-component which provides methods to manage queue data. It has two interfaces storer and retriever which respectively processes data to store and data to load.
- **DataAccessManager**: it is the sub-component which provides methods to manage data access. It has two interfaces storer and retriever which respectively processes data to store and data to load.
- **Retriever**: it is the sub-component which creates a query to get information based on the component which has request it.
- **Storer**: it is the sub-component which creates a query to store information based on the component which has request it.
- **Executor**: it is the sub-component involved in execute the query. It has two interfaces where the query comes from (load query and store query). The sub-component has also a directly interface with the DBMS

2.C Deployment View



In the deployment diagram it is shown the composition of the principal components of the system.

Smartphone: the user uses a mobile device to connect to the CLup service through the mobile application. The communication between the device and the servers is achieved through the HTTPS communication protocol.

QR scanner: these devices have the function of scanning the QR codes of the users entering and exiting the supermarket, also in this case the communication with the server is obtained through the HTTPS protocol.

Firewall: the firewall guarantees that access to the server is secure by blocking all connection attempts considered unsafe. This component also allows the server to be defended against possible cyber-attacks.

Load balancer: this component ensures that the computer traffic is equally distributed among the different parallel machines that compose the server.

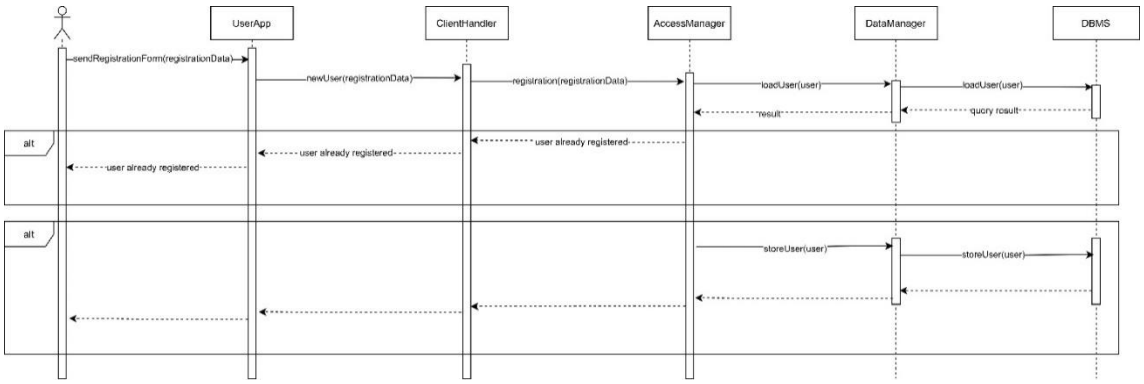
Supermarket Server: manages the connection of the various devices in the supermarket, the information received from them is processed and forwarded to the application server. We can therefore consider the supermarket server as a communication interface between the devices other than the smartphone and the application server. The supermarket server uses several physical machines in order to achieve better performance, moreover a possible failure, unless an unlikely simultaneous failure on all machines, does not severely compromise the functioning of the server.

Application server: This component contains the logic of operation of the entire system, given the thin client architecture, the user-side devices do not contain any part of the application logic. The application server correctly handles all requests coming from both the smartphone application and the supermarket server. The data necessary for processing information are saved in the data server, the interaction between these two servers is possible using the TCP / IP protocol. As already explained for the supermarket server, the application server is composed of several machines operating in parallel.

Data server: this component contains all the data necessary for the functioning of the system. On the data server operates the DBMS application, this deals with the management of requests to access to the database ensuring that there are no problems due to the use of multiple simultaneous transactions. The database is a relational type and in particular has been chosen as MySQL DBMS, the advantages of this choice are the low cost and the ease of development, and its wide distribution that makes it a standard given its compatibility with almost all systems.

2.D Runtime view

2.D.1 Registration Diagram

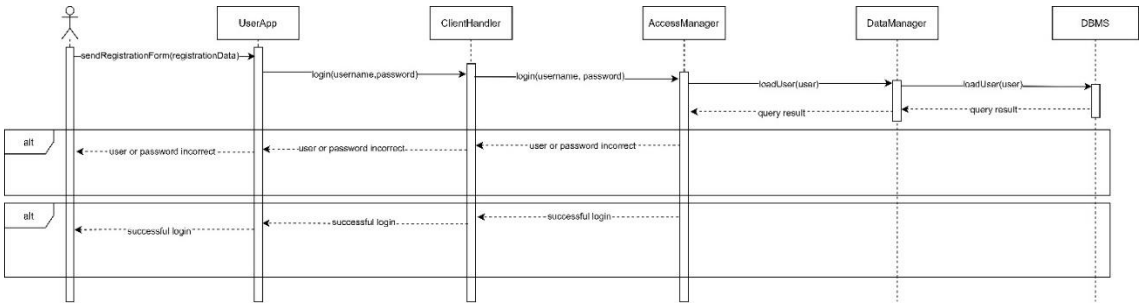


In this sequence diagram is represented how user interacts with CLup system during registration phase.

The user fills the registration form, giving all the information needed, on the registration page. UserApp component sends registration data to ClientHandler which forwards them to AccessManager. Once the registration data reach the AccessManager component, it creates a User object which contains registration data.

First of all DataManager component sends a request to check if an user with the same username already exists; if not, user credentials are saved in the database and in the other case the AccessManager component sends a request to the user informing him that the user is already registered.

2.D.2 Login Diagram

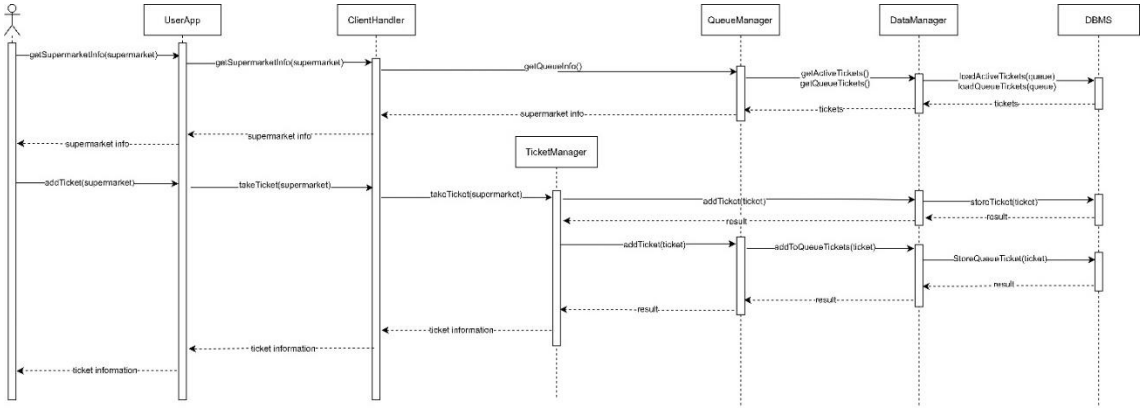


In this diagram is shown how user interacts with the CLup system during login phase

The user logs in with own username and password. ClientHandler receives the request and forwards it to the AccessManager component, DataManager makes a request to the database.

Once the result coming from the DBMS reaches the AccessManager and processed, If user or password are incorrect a notification is sent to the user, otherwise the AccessManager sends a positive ack to the user.

2.D.3 Take A Ticket Diagram



This diagram shows how the user interacts with CLup system when the user takes a ticket.

When the user selects the supermarket where he wants to take a ticket, a request of information of the selected supermarket is sent from the UserApp to ClientHandler component which then forwards it to QueueManager.

QueueManager component asks to DataManager to do a query to the DBMS to get information of the status of the queue of the supermarket, in particular the DBMS returns all the users who are waiting in the queue and all the people inside the supermarket. The information is then processed by the QueueManager which returns information about the supermarket, for example the number of users waiting in the queue and the waiting time estimated.

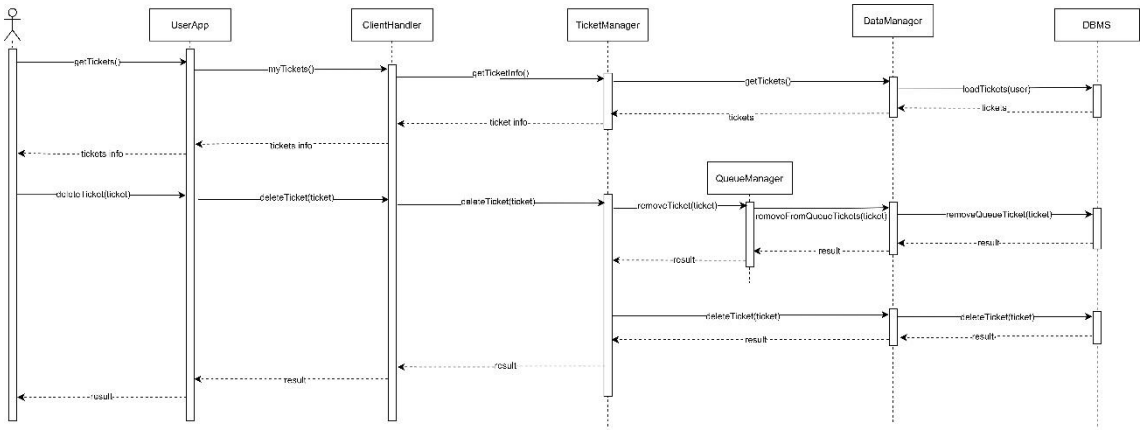
Once the user selects to take the ticket, a request is sent to the ClientHandler component which then forwards it to TicketManager.

The TicketManager sends a request to DataManager which makes a query to store the ticket in the system. The DBMS sends an ack to TicketManager.

TicketManager adds the ticket to the queue using TicketInsertion interface. The QueueManager makes a request to the DataManager which stores the ticket in the queue.

Finally an ack is sent to the user informing him that the ticket has been taken.

2.D.4 Delete A Ticket Diagram

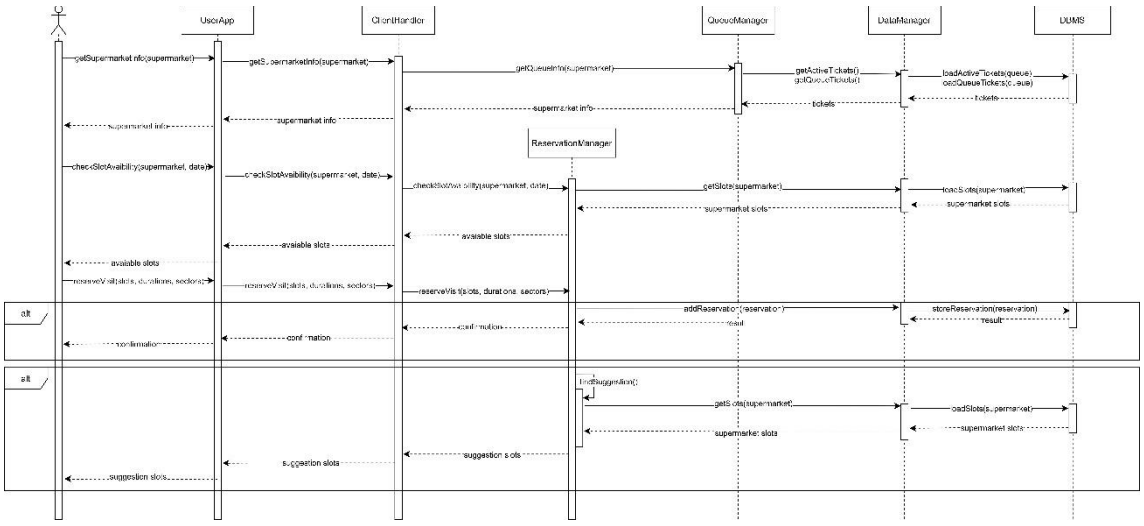


This diagram shows the interactions between the various components of the application server during the operation of deleting a ticket.

The procedure starts with a request from the user to view his generated tickets, then through the ticket management components (TicketManager and TicketData) a query is executed through the DBMS.

Once the ticket information is sent to the user, the user performs the deletion request for a specific ticket among those received. The ticket management components first delete the ticket from the queue. Finally, the ticket is removed from the database. The user is notified about the result of the operation.

2.D.5 Reservation



This diagram shows the interactions between the various components of the application server during the operation of reservation of a visit to the supermarket.

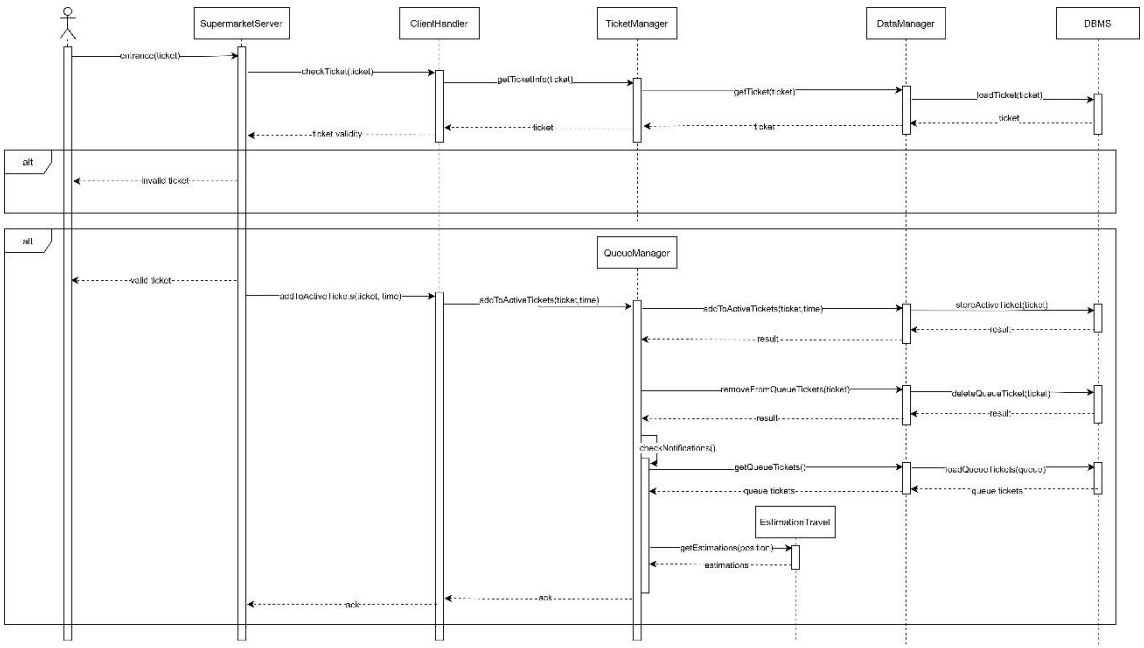
The user as first operation demands to the application server the information about a supermarket, the demand comes processed from the components of management of the queue (QueueManager and QueueData) executing one query in the database. Once the information about the supermarket is received, it is returned to the user.

The user then makes a request to view all available slots on a particular date. The request is handled by the ReservationManager component, which executes a query in the database to get information about the available slots.

Once all the information about the available slots has been obtained, the user processes the reservation request, eventually specifying the duration of the visit and the items he will buy. At this point the ReservationManager component checks that the requested reservation is correctly balanced with all other reservations already made in that supermarket. If so, the reservation is inserted into the database and confirmed to the customer.

If not, one or more alternative slots are identified within the same supermarket or in other supermarkets. These slots are then reported to the user, who will evaluate which one to choose and then reserve.

2.D.6 Supermarket entrance



This diagram shows the interaction between a QR scanner placed at the entrance of a supermarket and the application server, in particular all the interactions between components of the server are shown.

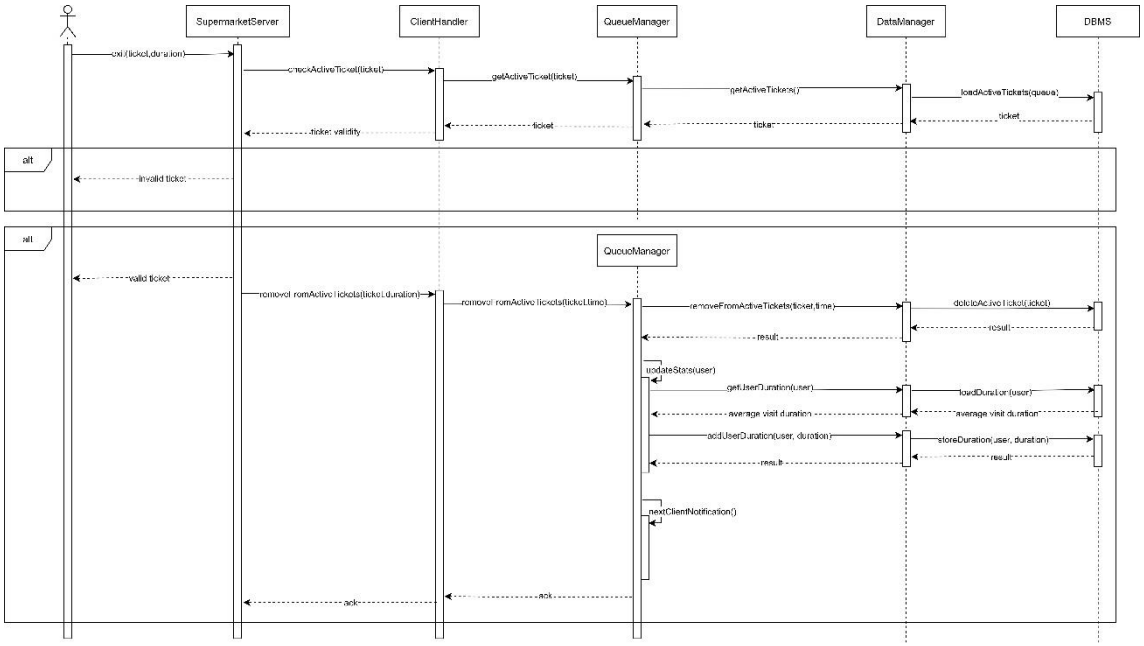
The procedure begins with the communication, from the scanner, of an entrance to the supermarket. This request is first processed by the SupermarketServer, then managed by the TicketManager which checks the correctness of the scanned ticket.

Once the SupermarketServer has been notified of the validity of the ticket, the insertion operation in the active tickets is performed. Through the QueueManager component, the

ticket is then inserted in the active tickets and removed from the queue tickets. Once the insertion operation is over, QueueManager checks all the notifications to be sent to the users in the queue, in order to establish when they are notified, an estimation of the time taken by the user to reach the supermarket is performed by the external component EstimationTravel.

Finally the result of the operation is notified to the SupermarketServer.

2.D.7 Supermarket exit



This diagram shows the interaction between a QR scanner placed at the exit of a supermarket and the application server, in particular all the interactions between components of the server are shown.

The procedure begins with the communication, from the scanner, of an entrance to the supermarket. This request is first processed by the SupermarketServer, then managed by the TicketManager which checks the correctness of the scanned ticket.

Once the SupermarketServer has been notified of the ticket validity, the ticket is removed from the queue. Through the QueueManager component the ticket is then removed from the active tickets. After the removal from the database, the system updates user's statistics about his average duration. Once the statistics are saved in the database, the QueueManager component notifies the next user in the queue that his turn has come. Finally the success of the operation is notified to the supermarket server.

2.E Component interfaces

In this diagram are shown all the interfaces of the system. For each interface some methods are shown which can be invoked from components. The methods pointed out in this diagram must not to be confused with the methods which will be coded during implementation. This diagram aims to give just an idea of what the components should use.

It's possible to identify five main categories of interfaces in the diagram.

In the first category there are MobileInterface and SupermarketInterface, both of them are used to forward requests to the right component of the system.

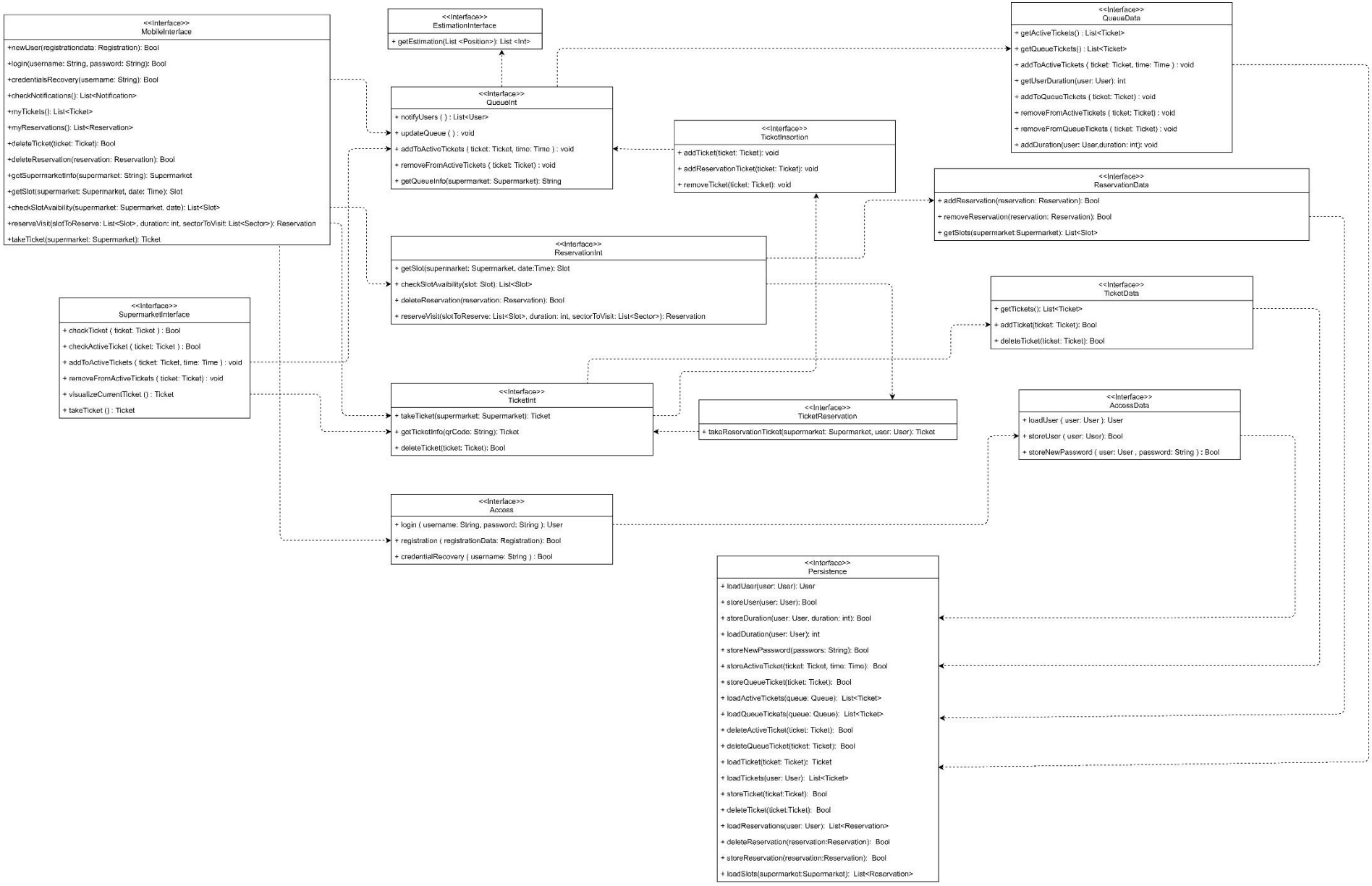
Mobile interface is useful because it allows the user to do direct requests to the system, using the mobile application. This interface, indeed, connects directly the UserApp component with the ClientHandler component. The supermarket is able to do requests thanks to SupermarketInterface which connects the SupermarketServer and the ClientHandler.

In the second category there are all the interfaces which have a direct relation with the client handler. Access, Ticket Int, ReservationInt and QueueInt are representatives of this category. These interfaces aim to handle specific requests of the user, each one of a specific request type.

In the third category there are all the interfaces which works as intermediary between components which are directly connected with ClientHandler. Some of them are interfaces between the component and an external API (RecoveryCredentialInterface, EstimationInterface), in the other cases between components which are directly connected with ClientHandler (TicketInsertion, TicketReservation)

In the fourth category there are all the interfaces used to prepare information to be stored or loaded: AccessData, TicketData, ReservationData, QueueData.

In the fifth category there is the Persistence interface which its main functionality is to collect all the information from DataManager component and to load or to store it acting as intermediary between component and DBMS.


<<Interface>>
MobileInterface
+newUser(registrationdata: Registration): Bool
+login(username: String, password: String): Bool
+credentialsRecovery(username: String): Bool
+checkNotifications(): List<Notification>
+myTickets(): List<Ticket>
+myReservations(): List<Reservation>
+deleteTicket(ticket: Ticket): Bool
+deleteReservation(reservation: Reservation): Bool
+getSupermarketInfo(supermarket: String): Supermarket
+getSlot(supermarket: Supermarket, date: Time): Slot
+checkSlotAvaibility(supermarket: Supermarket, date): List<Slot>
+reserveVisit(slotToReserve: List<Slot>, duration: int, sectorToVisit: List<Sector>): Reservation
+takeTicket(supermarket: Supermarket): Ticket
<<Interface>>
EstimationInterface
+ getEstimation(List <Position>): List <Int>
<<Interface>>
QueueData
+ getActiveTickets() : List<Ticket>
+ getQueueTickets() : List<Ticket>
+ addToActiveTickets (ticket: Ticket, time: Time) : void
+ getUserDuration(user: User): int
+ addToQueueTickets (ticket: Ticket) : void
+ removeFromActiveTickets (ticket: Ticket) : void
+ removeFromQueueTickets (ticket: Ticket) : void
+ addDuration(user: User,duration: int): void
<<Interface>>
QueueInt
+ notifyUsers ( ) : List<User>
+ updateQueue ( ) : void
+ addToActiveTickets (ticket: Ticket, time: Time) : void
+ removeFromActiveTickets (ticket: Ticket) : void
+ getQueueInfo(supermarket: Supermarket): String
<<Interface>>
TicketInsertion
+ addTicket(ticket: Ticket): void
+ addReservationTicket(ticket: Ticket): void
+ removeTicket(ticket: Ticket): void
<<Interface>>
ReservationData
+ addReservation(reservation: Reservation): Bool
+ removeReservation(reservation: Reservation): Bool
+ getSlots(supermarket:Supermarket): List<Slot>
<<Interface>>
ReservationInt
+ getSlot(supermarket: Supermarket, date:Time): Slot
+ checkSlotAvaibility(slot: Slot): List<Slot>
+ deleteReservation(reservation: Reservation): Bool
+ reserveVisit(slotToReserve: List<Slot>, duration: int, sectorToVisit: List<Sector>): Reservation
<<Interface>>
TicketData
+ getTickets(): List<Ticket>
+ addTicket(ticket: Ticket): Bool
+ deleteTicket(ticket: Ticket): Bool
<<Interface>>
SupermarketInterface
+ checkTicket (ticket: Ticket) : Bool
+ checkActiveTicket (ticket: Ticket) : Bool
+ addToActiveTickets (ticket: Ticket, time: Time) : void
+ removeFromActiveTickets (ticket: Ticket) : void
+ visualizeCurrentTicket () : Ticket
+ takeTicket () : Ticket
<<Interface>>
TicketInt
+ takeTicket(supermarket: Supermarket): Ticket
+ getTicketInfo(qrCode: String): Ticket
+ deleteTicket(ticket: Ticket): Bool
<<Interface>>
TicketReservation
+ takeReservationTicket(supermarket: Supermarket, user: User): Ticket
<<Interface>>
AccessData
+ loadUser (user: User): User
+ storeUser (user: User): Bool
+ storeNewPassword (user: User , password: String) : Bool
<<Interface>>
Access
+ login (username: String, password: String): User
+ registration (registrationData: Registration): Bool
+ credentialRecovery (username: String) : Bool
<<Interface>>
Persistence
+ loadUser(user: User): User
+ storeUser(user: User): Bool
+ storeDuration(user: User, duration: int): Bool
+ loadDuration(user: User): int
+ storeNewPassword(passwors: String): Bool
+ storeActiveTicket(ticket: Ticket, time: Time): Bool
+ storeQueueTicket(ticket: Ticket): Bool
+ loadActiveTickets(queue: Queue): List<Ticket>
+ loadQueueTickets(queue: Queue): List<Ticket>
+ deleteActiveTicket(ticket: Ticket): Bool
+ deleteQueueTicket(ticket: Ticket): Bool
+ loadTicket(ticket: Ticket): Ticket
+ loadTickets(user: User): List<Ticket>
+ storeTicket(ticket:Ticket): Bool
+ deleteTicket(ticket:Ticket): Bool
+ loadReservations(user: User): List<Reservation>
+ deleteReservation(reservation:Reservation): Bool
+ storeReservation(reservation:Reservation): Bool
+ loadSlots(supermarket:Supermarket): List<Reservation>

2.F Selected architectural styles and patterns

We decided to use some patterns commonly used in software engineering, among these we used:

- Client-server pattern: it's a pattern where there is a provider of service, called server and a service requesters called client. This architecture let the system store information in a central location and it guarantees more security.

- Three Tiers pattern: the system is organized into three logical and physical computing tiers:
    - Presentation
    - Business Logic
    - Data

  The Three Tiers pattern choice is made because it brings some important benefits, as we described in a detailed way in the Overview section of this document:
    - Scalability
    - Security
    - Reliability
    - Faster development

- Cache: We decided to introduce a cache. With this mechanism data are stored for future requests. The business logic is less busy and the user can be served faster.

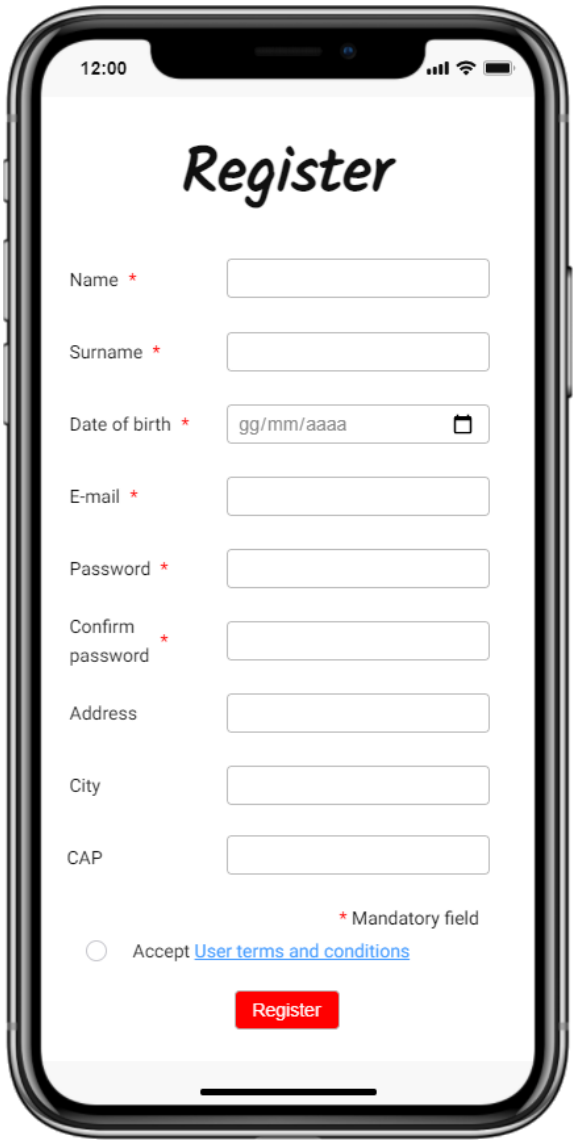
- Thin Client pattern: it allows the user to have a simple logic application on their devices. The choices of this pattern are described in a detailed way in the Overview section.

- https: Requests from client to server are made using https protocol. We decided to use https instead of https because of some extra functionalities which we consider important as data encryption, data protection, server authentication. Nevertheless there are some disadvantages as extra overed and TLS certificate cost.

- MVC: We decide to user Model View Controller pattern because in this way the view is separated from the Model and it is possible to update graphics interface independently from the other two parts. From the practical point of view it is easy to design a similar architecture with three tiers pattern.

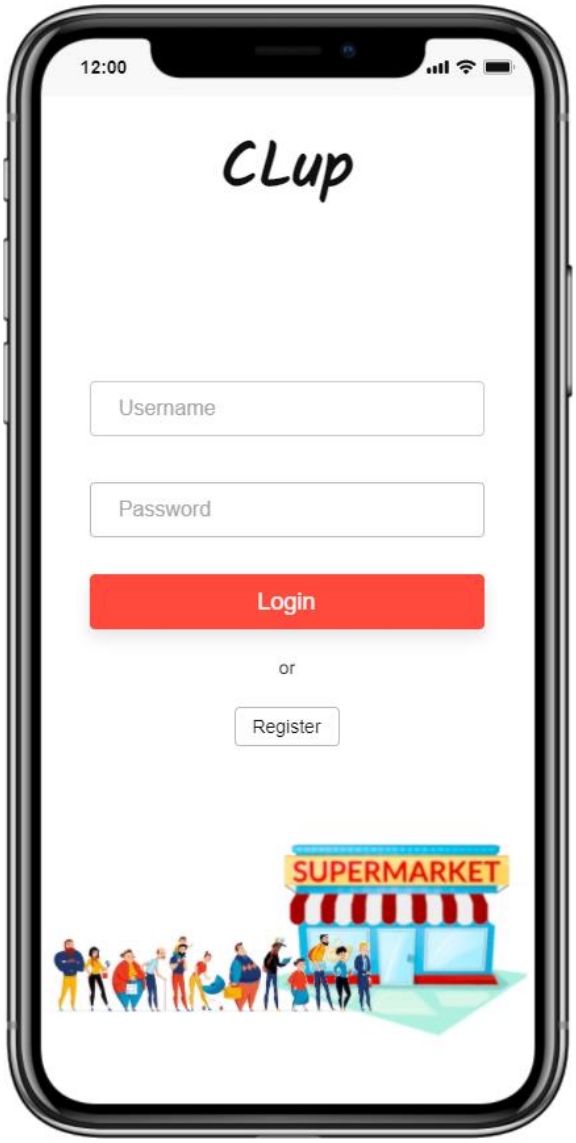
- Façade: This pattern it's useful to mask complex structures of the system with interfaces which makes interaction easier.

3. User Interface Design



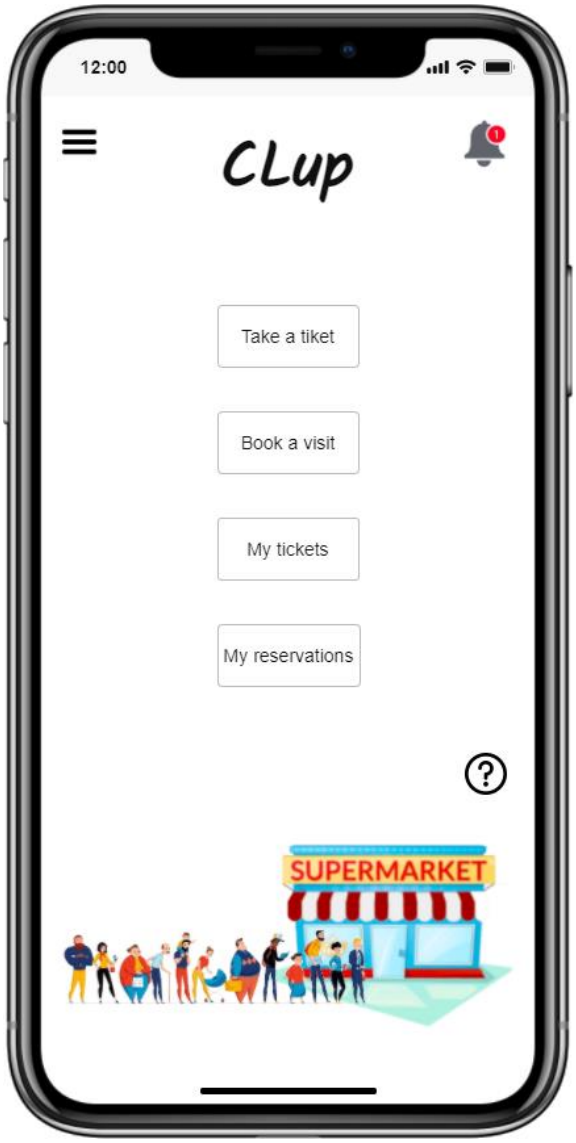
Registration Page

This mockup shows the Registration page. In this page the user can register to CLup system providing some personal data.



Login Page

This mockup represents the login page. In this page the user is asked for username and password prevously provided during registration.

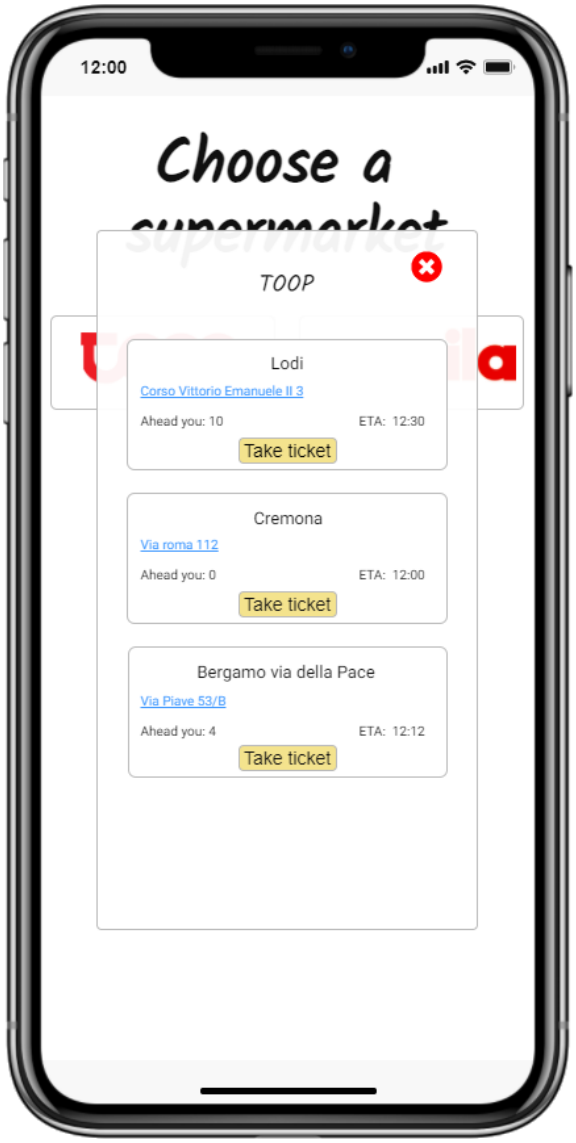


Home Page

This mockup represents the home page of application. This page aims to facilitate navigation to other pages and provides the functionalities of the system.

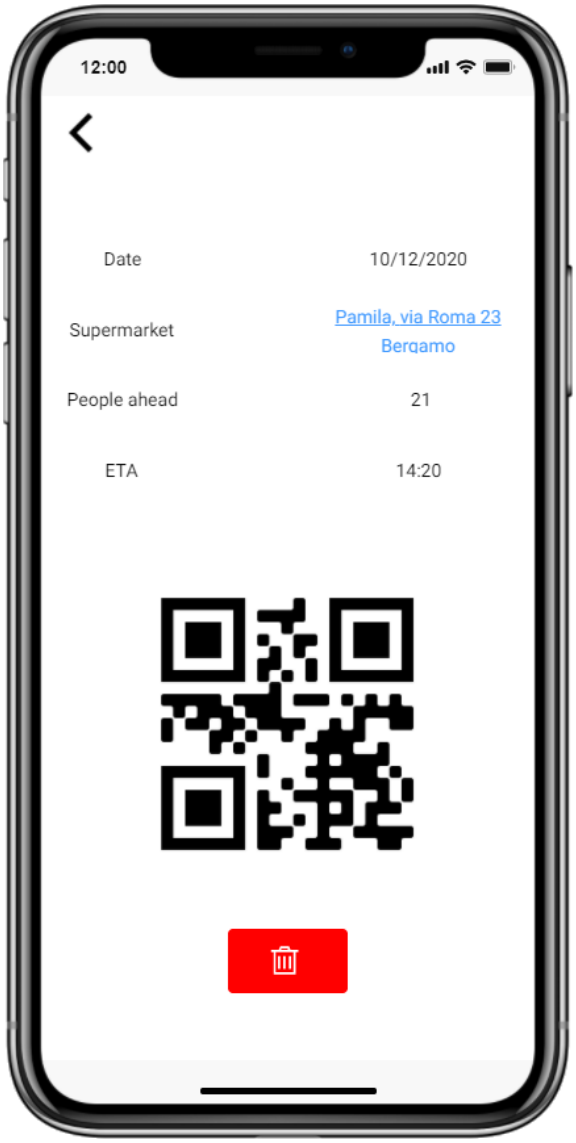
Choose Supermarket Page

When “Take a ticket button” is pushed, this page is shown. In this page the user can choose the supermarket in which he wants to purchase.



Take Ticket Page

Once the user has selected the supermarket, this page is displayed. In this page the user can take a ticket from one of the physical supermarket available.

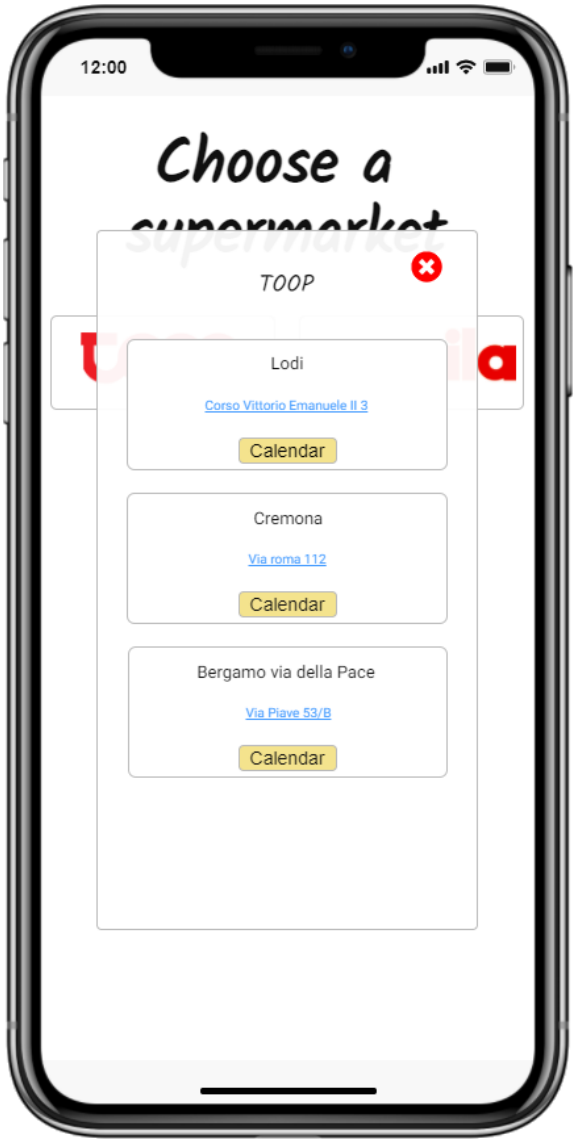


Ticket Page

In this page is displayed the ticket with some information regarding queue and supermarket information. The ticket has a QR wich the user must scan at the entrance and the exit of the supermarket. There is also a button to delete the ticket.

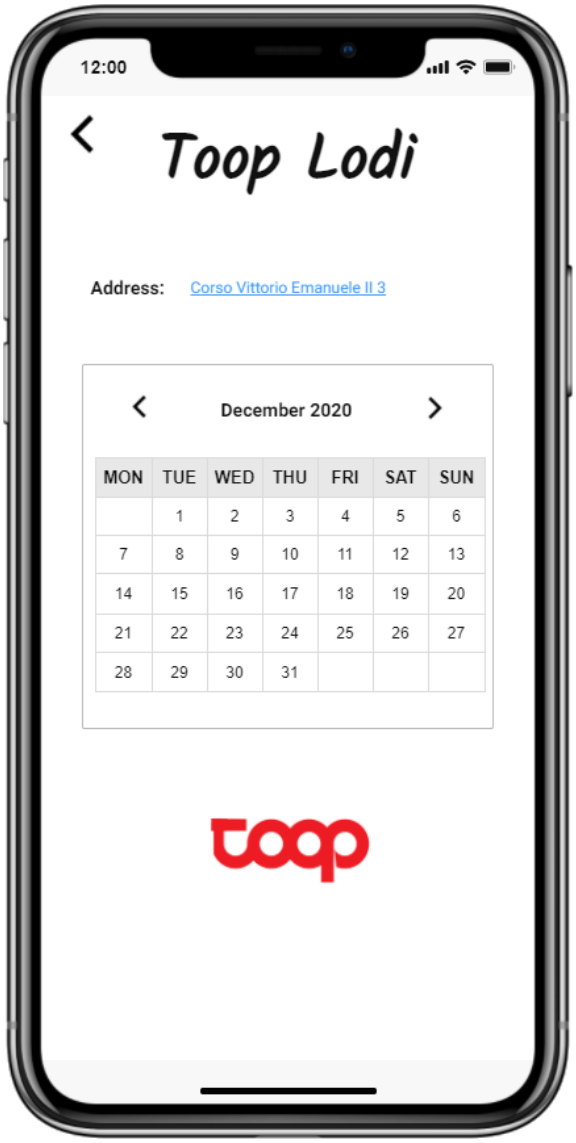
Choose Supermarket Page

When “Book a visit” button is pushed, this page is shown. In this page the user can choose the supermarket in which he wants to purchase.



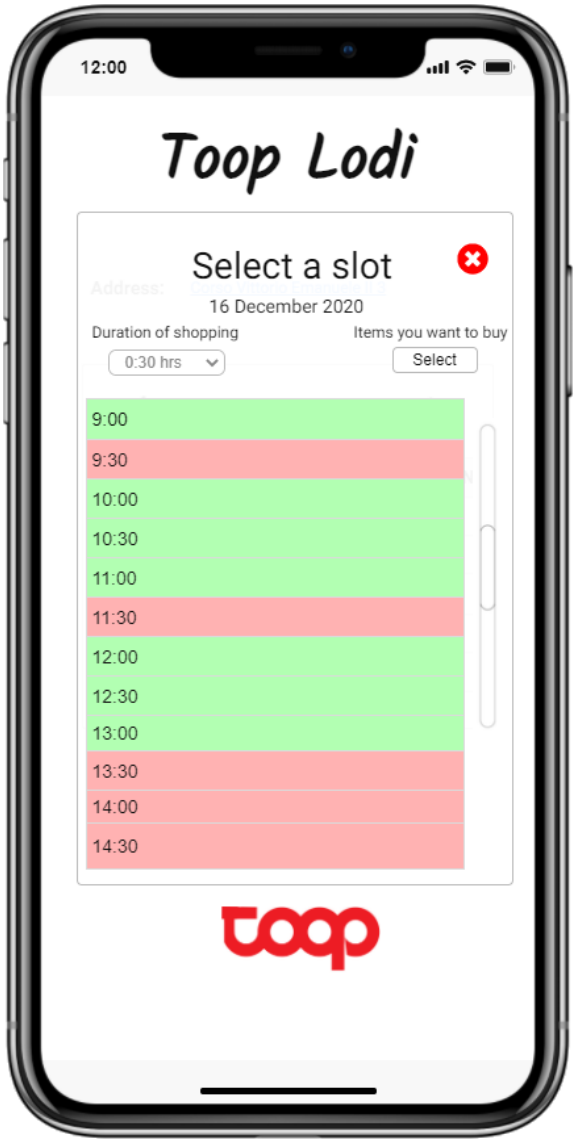
Book Visit Page

Once the supermarket is selected in order to book a visit, this page is displayed. In this page the user can select one of the physical supermarket available. Finally The button “Calendar” let the user select a slot.



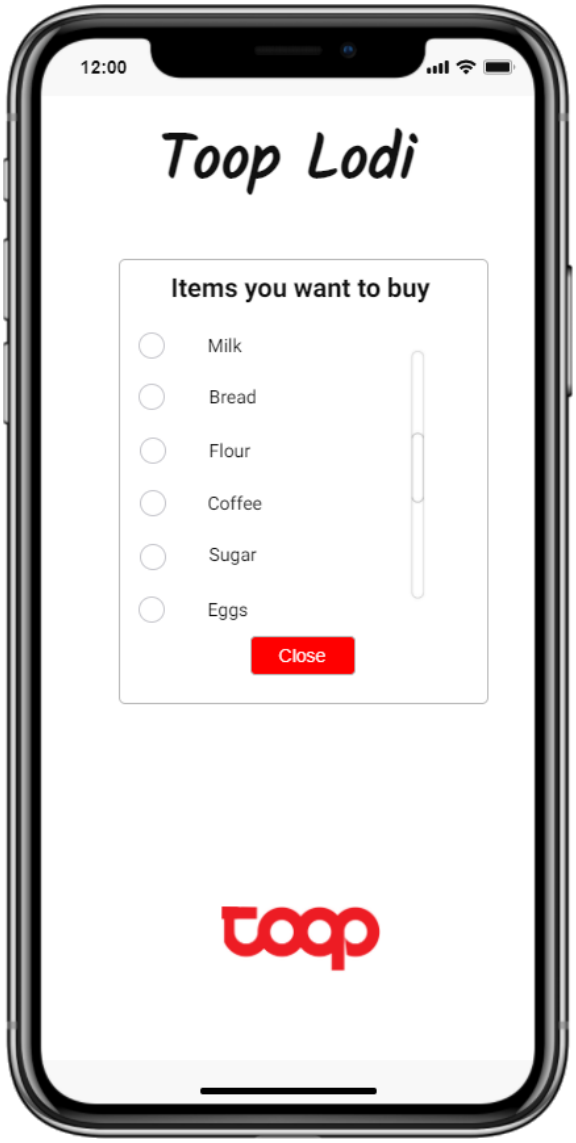
Supermarket Calendar

This screen belongs to the booking process, it is shown a calendar from which the user can choose the date on which he wants to make the reservation.



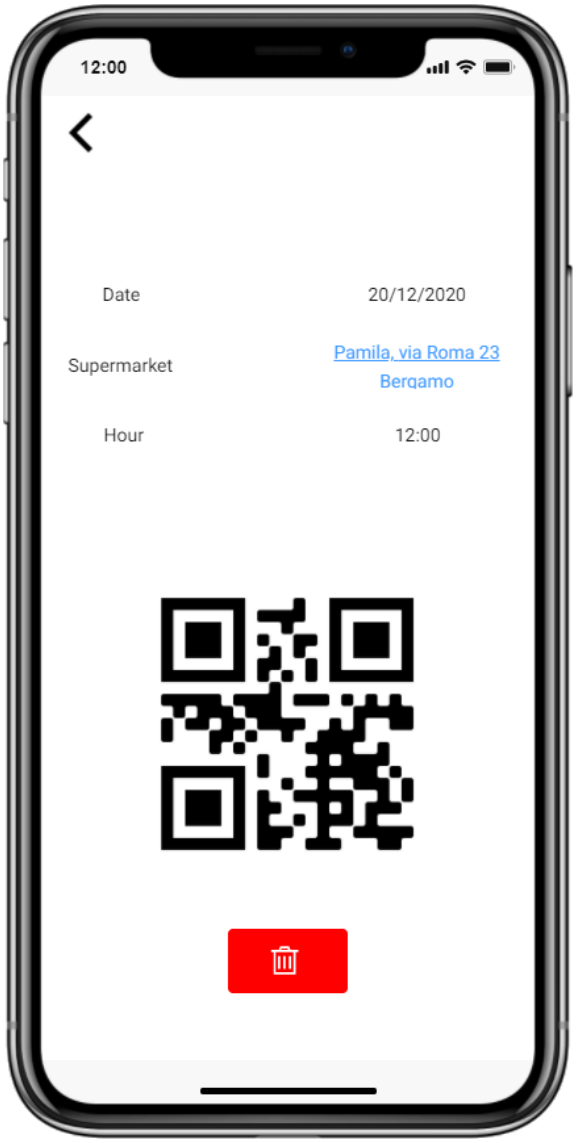
Slot availability

This screen belongs to the booking process, it is displayed the availability of slots for the selected date and supermarket. The user can also add the duration of his visit and select the products he is planning to buy.



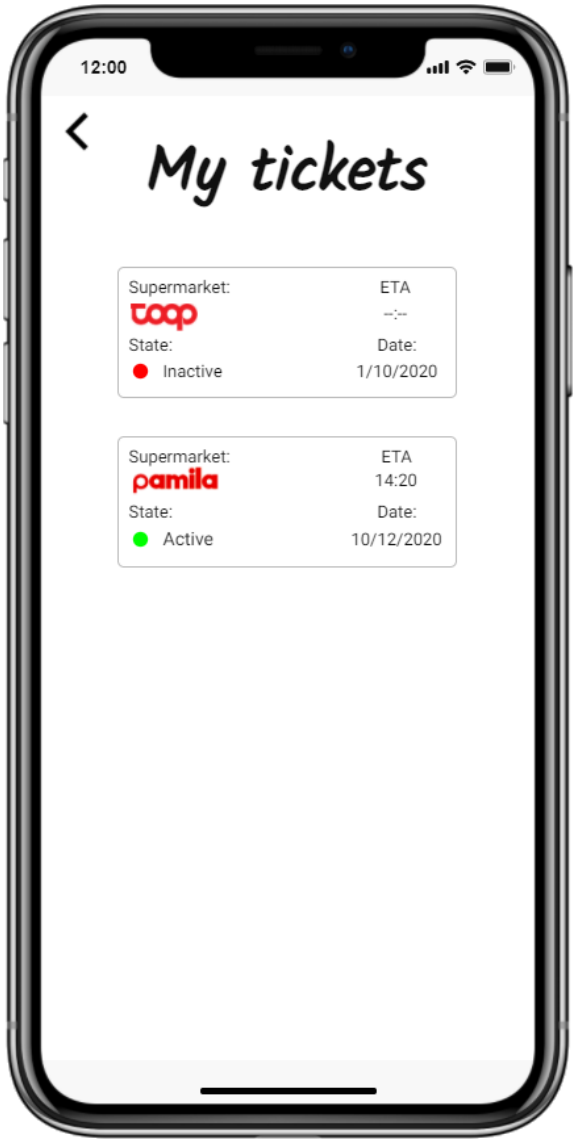
Items to buy

This screen belongs to the booking process, the user can add the products he intends to buy.



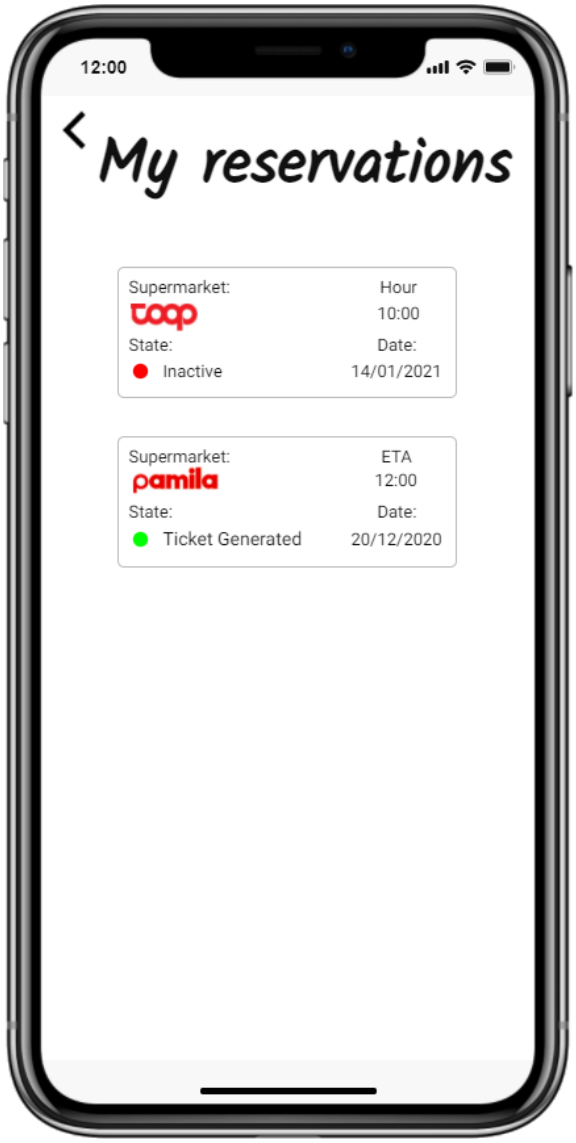
Reservation

This screen displays the details of a reservation made by the user. If this is active, the QR code generated will also appear. There is also a button to delete the reservation.



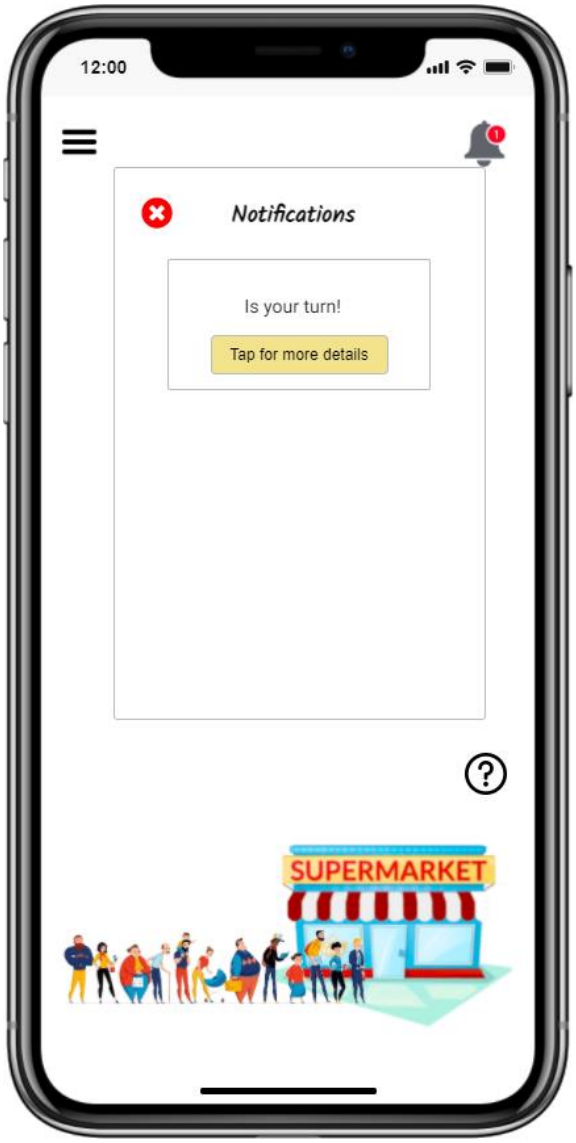
My Tickets

This screen shows tickets taken by the user. A ticket can be active or inactive if it is just expired its turn.



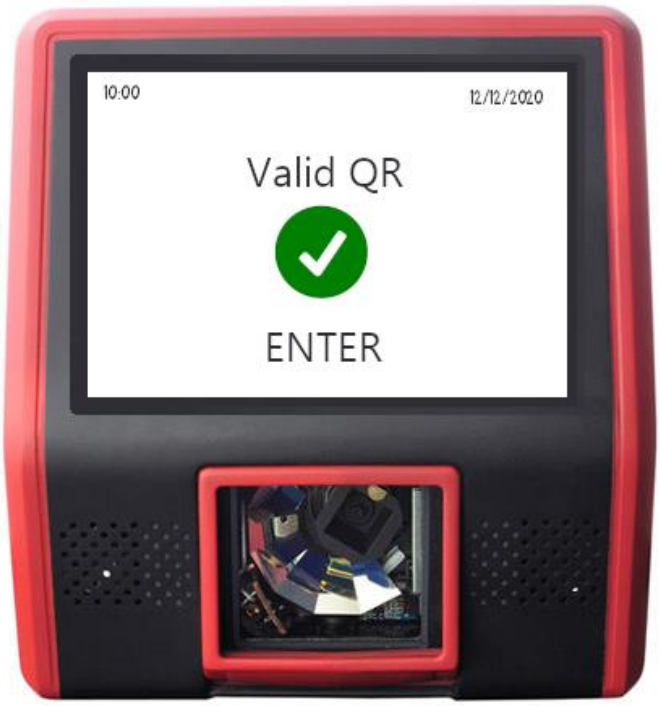
My Reservations

This screen shows the user the reservations made by the user. The reservation can be active or inactive, in case it is active the QR code of the reservation has been generated.



Notification Menu

This screen shows the appearance of the notifications menu. When a notification arrives, an entry corresponding to the received notification appears in this menu.



QR Scanner

This mockup refers to the QR scanner present at both the entrance and exit of each supermarket. Once the code is scanned, the screen will show the result of the operation

4. Requirements Traceability

R1	The application must consider the tickets taken on the spot	TicketManager
R2	The application creates a virtual queue for customers	QueueManager TicketManager
R3	The application tracks the entrance of customers	QueueManager
R4	The application tracks the exit of customers	QueueManager
R5	User indicates the items that will be purchased	ReservationManager
R6	User indicates the approximate duration of the visit	ReservationManager
R7	The application must assign a ticket to each user asking for it	TicketManager
R8	The application let the user to select a date in a calendar to allow him to book a visit	ReservationManager
R9	The application estimates the time needed to reach the supermarket	QueueManager
R10	The application tracks the number of customers who booked a slot	ReservationManager
R11	The application knows the number of customers inside all the supermarkets of the chain	DataManager
R12	The application notifies the customers that their turn is coming	QueueManager

R13	The application notifies the customer that his ticket expires	QueueManager
R14	The application advices the customer to choose a different slot if the selected one is occupied	ReservationManager
R15	The application advices the customer to choose a different slot if the selected one is more crowded	ReservationManager
R16	The application estimates the duration of a visit of customers and store the information in a database	DataManager QueueManager
R17	The user can log in	AccessManager
R18	The user can register himself in the application	AccessManager

5. Implementation, Integration and Test Plan

In this section we explain the development process, we provide a plan of how the development team will operate to finally realize the project as it has been described in this document.

We describe the motivations of the development approach we have selected and we analyze the order in which each component will be developed.

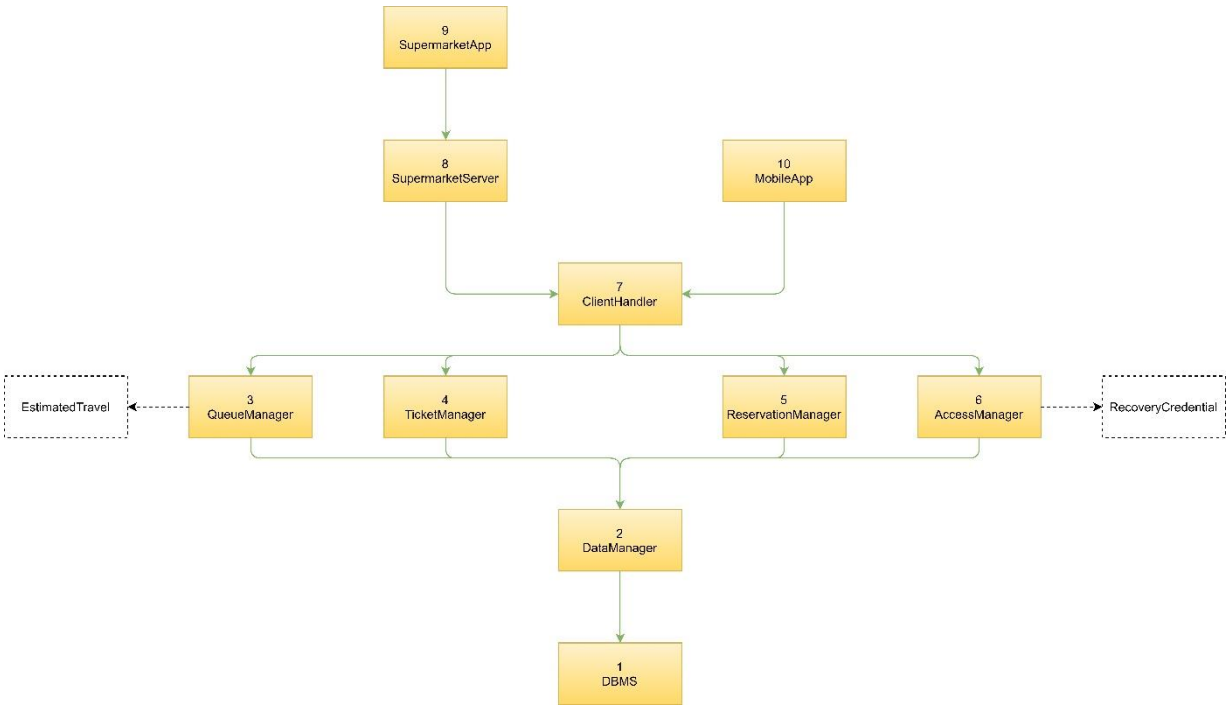
Finally we provide a test plan of how we proceed with unit testing and integration testing

5.A Implementation Plan

We decided to use a mixed approach between Top-Down and Bottom-Up, in particular we chose to use Top-Down approach to breaks the system structure into its components and then a Bottom-Up approach to develop each single component.

Top-Down approach gives us some advantages as the “divide et impera” approach which let us divide a problem in multiple easier parts.

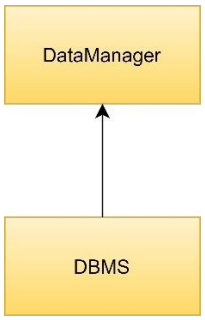
Bottom-Up approach gives us many advantages when we are dealing with the single component, in particular we can reuse low-level utilities already implemented ant it also let development team reuse code. Moreover with a Bottom-Up approach is easier develop components in a parallel way reducing development time.



In the previous image we show all the components we have considered using a Top-Down approach. We also represent the components in relation with the order on which they will be implemented using a Bottom-Up approach.

1. The first component to implement is the DBMS

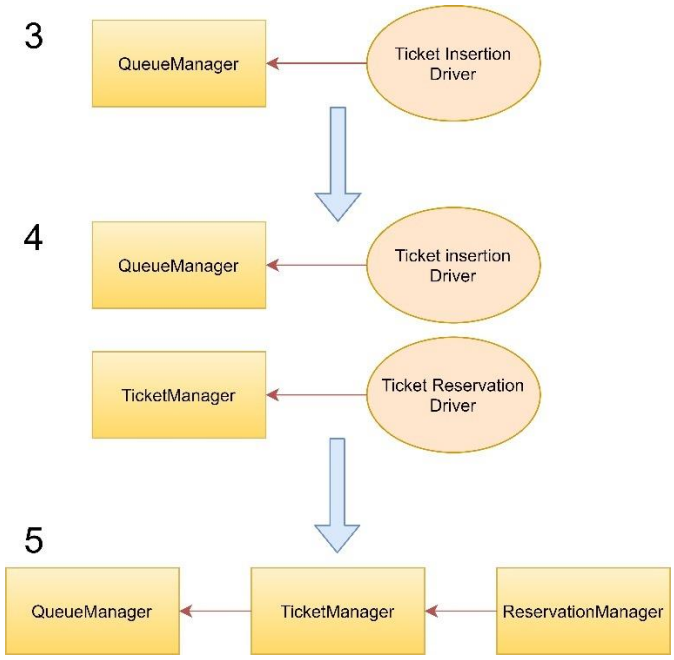
2. The second component to implement is the DataManager which is in charge to load and store information in the database. Manager components use DataManager. Once we have implemented the DataManager component, we will integrate both components and will test them.



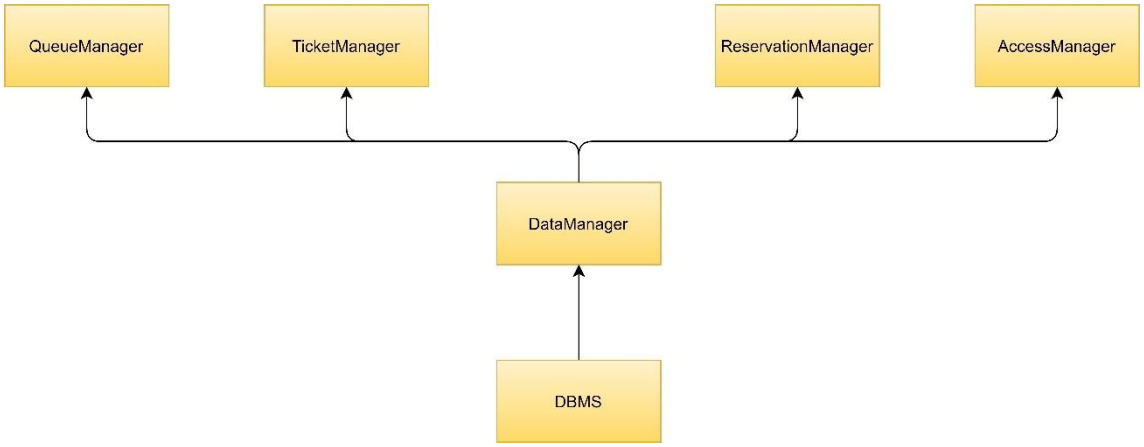
3. While testing the previous components, it's possible to start implementing QueueManager component which has the main functionalities of the application. This component is almost independent from the others except for some functionalities dealing with ticket insertion.
   For those reason we decide to use a driver in charge to call the methods contained in the TicketInsertion interface.
   This component handles interaction with an external interface which is not part of the system. Those methods are the last to be implemented and tested because of the clear documentation of the API

4. Once QueueManager component is implemented, TicketManager is the next one. The TicketManager is also independent from the others components except for some functionalities dealing with ticket taken from reservation.
   As done before it is possible to use a driver which call the methods contained in TicketReservation interface.

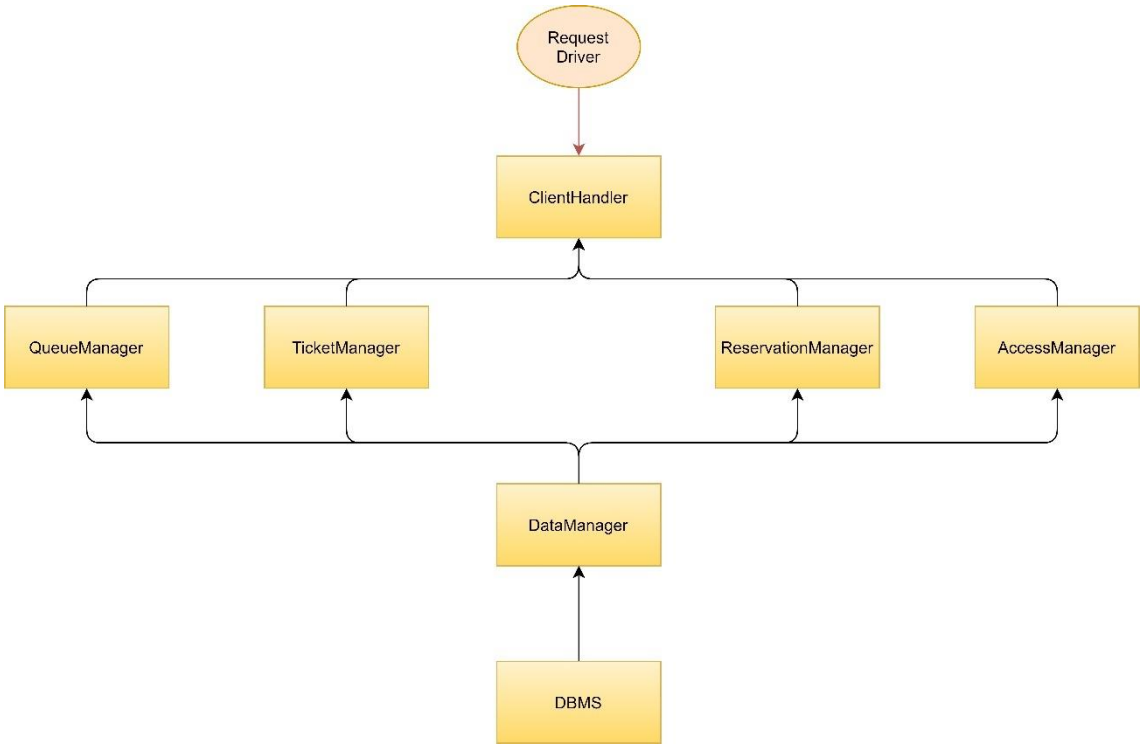
5. The fifth component to implement is ReservationManager. Once the implementation and unit test are completed, all the driver used until now are removed and an integration test is made on QueueManager, TicketManager and ReservationManager.



6. The next step is to implement the AccessManager which is completely independent from the others manager components, for this reason a driver is not needed. This component handles interaction with an external interface which is not part of the system. Those methods are the last to be implemented and tested because of the clear documentation of the API. Once this component is implemented and tested, an integration test is made between all the components implemented until now.



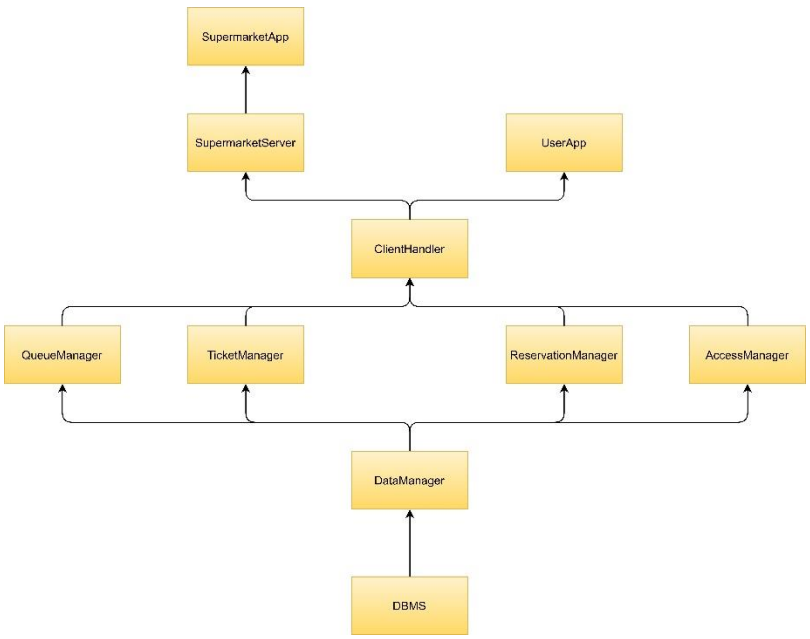
7. Since the components are now fully tested, it is possible to implement the ClientHandler. This component handles requests coming from both the user and the supermarket. We decide to use a driver behaving as mobileApp or SupermarketServer to call methods of the client handler. In this way we could test our system independently from client application.



8. Once the system is tested it's possible to start implementing the SupermarketServer component.

9. Then the supermarket application is developed and tested

10. As the last step there is the implementation of the UserApp, this component can also be implemented concurrently with the SupermarketServer component.
    Once all the components are implemented and tested, the entire system is tested with the last integration test.



5.B Testing

The implementation of the system will be tested at different times during the development process. In particular, for each implemented component, a unit test will be performed on it, where it will be checked that the written code reflects the operation prescribed to that component. Since the development process adopted involves the bottom-up approach, it is necessary to use drivers for unit testing. The unit test will be performed with the white-box testing approach, in fact the internal structure of the component is known and the quality of the written code can be checked. The techniques used for unit testing will be: statement coverage, branch coverage and condition coverage, the goal of testing is to have a high coverage for each of these conditions. Once different components have been developed and tested individually, we can proceed with different approaches to their integration: incremental integration or big bang integration. In our case the incremental approach was chosen, this has the advantage of allowing developers to understand the source of any errors and simplifies the correction. Integration testing is performed with the principle of black-box testing, in fact developers do not consider the content of individual components, but the interaction between them and the correction of the final result. It should be noted that, proceeding with the integration of individual components, the presence of drivers is no longer necessary, as they are replaced by the developed units.

Once all components are integrated the whole system will be tested, in particular it is necessary to check that the final product actually meets all functional and non-functional requirements. This type of testing aims to ensure that the system is suitable for the use for which it was designed, so it must be carried out in a condition as similar as possible to that which will be presented once in operation. The system testing can be divided into different types:

- Functional testing: with this type of test it is checked that the system behaves correctly from the functional point of view, as described by the functional requirements. As for the integration test is adopted a black-box methodology, we are not interested in the composition of the system but only in the results that it produces against certain inputs. In particular, the model-based testing approach is used to identify the cases to be tested.

- Performance testing: with this type of testing the performance of the system is checked, to perform it is necessary to know the expected load of the system and what are the performances considered acceptable. Once the testing is done, we improve eventual bottleneck and we try to correct inefficient algorithms.

- Load testing: this type of testing aims to identify any bugs due to high system load, typically buffer overflows and memory leaks. This test can be useful to identify the maximum load that the system can bear for a long period of time.

- Stress testing: in this case is checked that the system is able to recover the normal operation after a failure or an unusual event.

5.C Additional Specification

For what concerns the development tools, we decide to use Java as the main language of the system because of its popularity and its interoperability between different platforms. Java is also used for the supermarket application.

Regarding the DBMS, we decide to use SQL as relational database management language.

For the development of the user application we decided to use Flutter, an open-source UI software development kit created by Google. The interested thing of Flutter is that it is possible to create from the Dart code both Android and iOS application. In addition to this, it's very easy to use and it has numerous open source packages which can help customize the graphics of the application.

For what concerns testing we use JUnit and Mockito.

6. Effort spent

6.B Alberto Maggioli effort

Task	Hours
Component View	3 h
Deployment View	1 h
Runtime View	5 h
User Interface Design	8 h
Testing	2 h
Total	19 h

6.B Lorenzo Poretti effort

Task	Hours
Introduction	1,5 h
Component View	5 h
Runtime View	3 h
Component Interfaces	1 h
Selected Architectural Styles and Patterns	2 h
Implementation Plan	5 h
Additional specification	1 h
Total	18,5 h

6.B Team effort

Task	Hours
Component View	10 h
Deployment View	2 h
Runtime View	1 h
Component Interfaces	6 h
Requirement traceability	1 h
Implementation Plan	3 h
Testing	1 h
Additional specification	0,5 h
Total	24,5 h

7. References

[1] Alloy Documentation: http://alloytools.org/documentation.html

[2] Slides of the lectures of “Software Engineering II” course

[3] Book: “Software engineering principles and practice”, Hans Van Vliet

[4] Diagrams: made with Draw.io: https://app.diagrams.net

[5] Screens: https://mockittapp.wondershare.com

[6] “Requirement Analysis and Specifications Document CLup”, Alberto Maggioli, Lorenzo Poretti