Politecnico di Milano

A.A. 2020/2021

POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2

DD
Design Document
CLup: Customers Line-up

Group members:

Andrea Razza (968483)

Loris Panza (967915)

Version: 1.0

Chapter 1

Introduction

1.1 Purpose

Here a brief review of the purposes of the application CLup that were already identified in the RASD document.

The main purpose of this application is to manage client's accesses into shops in a pandemic context caused by Covid-19. Specifically, the application allows the user to avoid creating crowds and queues at stores entrance and improve the ways to visit them. Specifically, the system allows to avoid wasting and waiting time in queues giving the possibility to book an access to the shop indicating the necessary information. In addition, CLup is an advantage also for the shop owners: it helps to manage the clients' influx inside and outside the shop and allows the respecting of the social distancing policy between customers inside the store. Therefore, the customer can visit shops in safe condition and the shops can continue to sell their products without caring about the number of people inside the shop.

The purpose of the Design Document is to give more technical details tha te RASD. The latter has the objective to provide a more abstract view of the system with its functionalities while the Design Document describes the components of the system, their interactions and design choices which will be useful in the implementation phase.

1.2 Scope

Here a brief review of the system's scopes which has been already stated in the RASD document.

CLup offers two different services:

- Basic service: the system allows users to sign up and wait their turn from home visualizing how many people precede him. Every customer in the queue has a unique QRcode. The system should provide customers with a reasonably precise estimation of the waiting time and should alert them taking into account the time they need to get to the shop from the place they currently are. This place can be specified in two ways: either through GPS or indicating a specific address. In order to manage in an optimal way the queue waiting time, every user can specify an approximate expected duration of the visit.
- Booking a visit: users can also submit a reservation indicating the same information of the basic service, except the position, plus the specific time and date. The customer can insert

various types of products that he is going to purchase in order to allow the software to coordinate optimally and, eventually, contemporary clients in the market.

In order to be available in the application, the grocery shop should sign up to the software indicating specific information.

For people that have no access to this technology there are ticket machines which physically distribute tickets. The customer can queue thanks to these tickets: specifically, the user will indicate the required information to the grocery's ticket machine. This machine, acting as a proxy, will send the information to the software and will print automatically a ticket where the user can visualize all the details of the required service. In this way, CLup is constantly updated in order to manage both types of clients (physical and online ones).
It is necessary also to distinguish physical and virtual interactions: for instance, the application should be aware of sending notifications only for digital users and not for the physical ones.
If the user doesn't specify the expected time, CLup infer it from an analysis of the client's previous visits in the shop he has specified in the request. If the client has never gone to that grocery or has never indicated any expected period, the software estimates an approximate period based on the number of people in the queue and the shop's dimension.
When a customer arrives at the market, the QR code generated by the application or printed on physical tickets must be scanned to make sure that the person is allowed to enter, through reservations or queue. This scan is useful also to understand and store the effective time spent in the shop. Specifically the QR code has to be scanned when the user enters the shop and when he leaves it to define the effective time.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Queue	Sequence of clients that wait their turn keeping distance from the shop.
Digital client	User who takes advantages from services offered by the application through the digital software.
Physical client	Users who have no possibility to use the software and go physically to the grocery store to access

	the application service.
Ticket	Certification for physical clients of having benefited the service.
Client scanning operation	QR code scanning upon client arrival at the shop.
Alert / Notification	Message sent by CLup app to alert the customer of going to the shop.
Estimated time	Time calculated by the application if the user does not insert any expected time.
Expected time	Time specified by the user that indicates how long that customer will stay in the shop.
Remaining time	Time calculated by the application and visualizable by a digital user that indicates how much time is remaining before being into the shop.
Effective Time	Effective time passed between the entrance of the shops and the exit.
To line up / To queue/ To join the queue	Being part of a virtual queue and waiting for the turn.
Overflow user	A booked user who is allowed to enter a full shop because it has indicated in his request a type of product different from all the other simultaneous booked clients in the shop.

1.3.2 Acronyms

- QR Code: Quick Response Code;
- RASD: Requirement Analysis and Specification Document;
- GPS: Global position system.
- HTTPS: HyperText Transfer Protocol over Secure Socket Layer

1.3.3 Abbreviations

- Gn: n-th goal;
- Dn: n-th domain assumption;
- Rn: n-th requirement;
- WPn: n-th world phenomena;
- SPn: n-th shared phenomena.

1.4 Reference Documents

- Specification document: “R&DD Assignment AY 2020-2021”;
- Slides from the lectures.
- RASD Document

1.5 Document Structure

- Chapter 1 describes the scope and purpose of the DD, including the structure of the document and the set of definitions, acronyms and abbreviations used.
- Chapter 2 contains the architectural design choice, it includes all the components, the interfaces, the technologies (both hardware and software) used for the development of the application. It also includes the main functions of the interfaces and the processes in which they are utilised (Runtime view and component interfaces). Finally, there is the explanation of the architectural patterns chosen with the other design decisions.
- Chapter 3 shows how the user interface should be on the mobile and web application.
- Chapter 4 describes the connection between the RASD and the DD, showing the matching between the goals and requirements described previously with the elements which compose the architecture of the application.
- Chapter 5 traces a plan for the development of components to maximize the efficiency of the developer team and the quality controls team. It is divided in two sections: implementation and integration. It also includes the testing strategy.
- Chapter 6 shows the effort spent for each member of the group.
- Chapter 7 includes the reference documents.

Chapter 2

Architectural design

2.1 Overview: High-level components and their interaction

The application to be developed is a distributed application structured according three logical layer:

- **Presentation Layer**: the presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one that allows the interaction between users and system and which displays content and information useful to an end user.
- **Application Layer**: the application tier contains the functional business logic which drives an application's core functionalities.
- **Data Access Layer**: the data tier comprises the database/data storage system and data access layer.

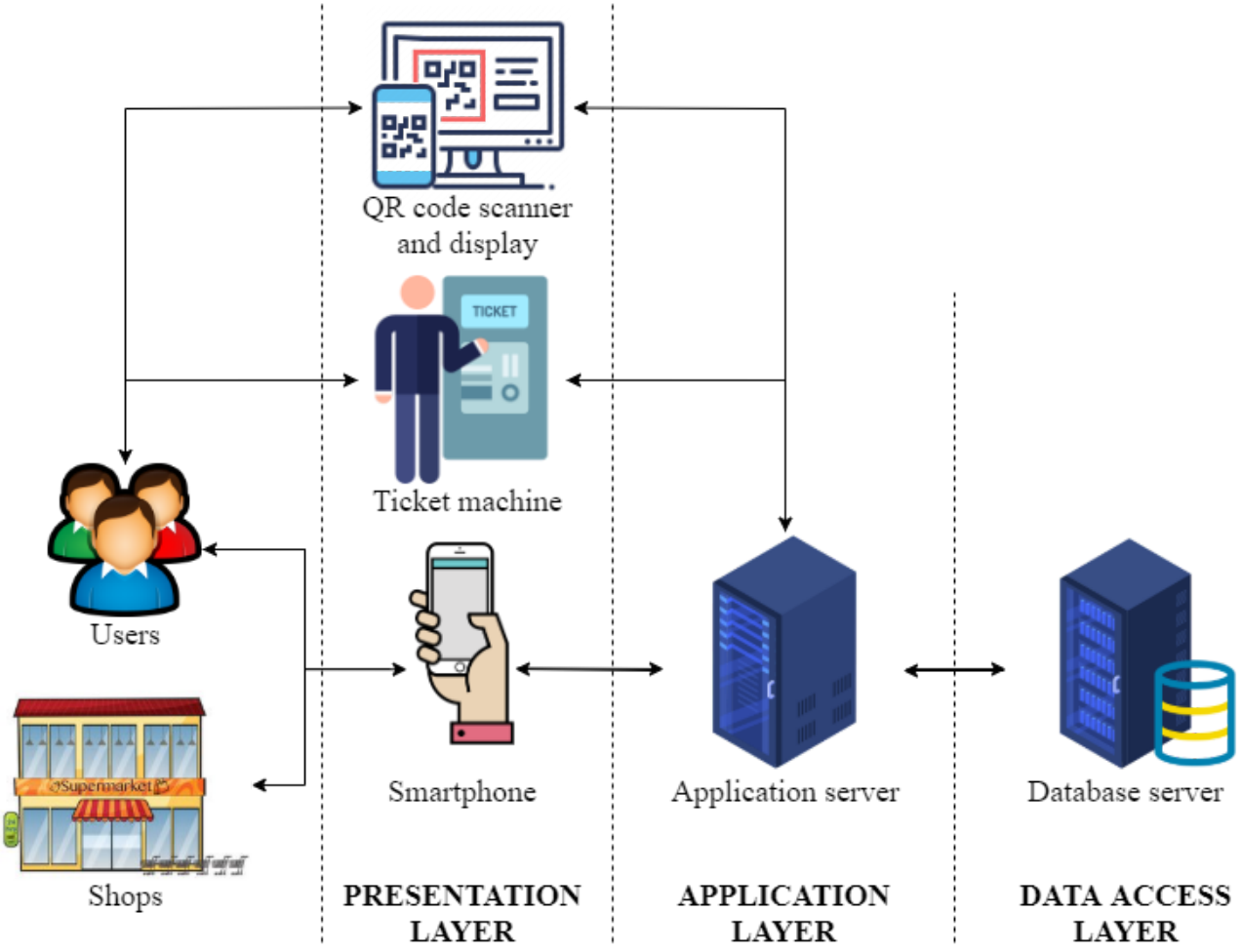
By separating out the different layers it is possible to scale each independently depending on the need at any given time. This permit to load balance each layer independently, improving overall performance with minimal resources. Additionally, the independence created from modularizing the different tiers gives many deployment options. It is possible to increase reliability and availability by hosting different parts of the application on different servers and utilizing cached results. With a full stack system, the developers does not have to worry about a server going down and greatly affecting performance throughout the entire system, but with a 3-layer application, the increased independence created when physically separating different parts of an application minimizes performance issues when a server goes down. Having an intermediate layer between client and server can guarantee more security to the access control of the database from the users.

Either shop or digital user interacts with the application server to register to CLup and to log in it.

- **Basic Service**: a user wants to join a queue and sends the queue request to the application server, the server receives it, checks it and, if this operation is successful, stores the request in the database. This operation can be completed using a ticket machine or a smartphone.
- **Booking service**: a digital user wants to book a visit and sends the request to the application server, the server receives it, checks it and, if the operation is successful, stores the booking request in the database.

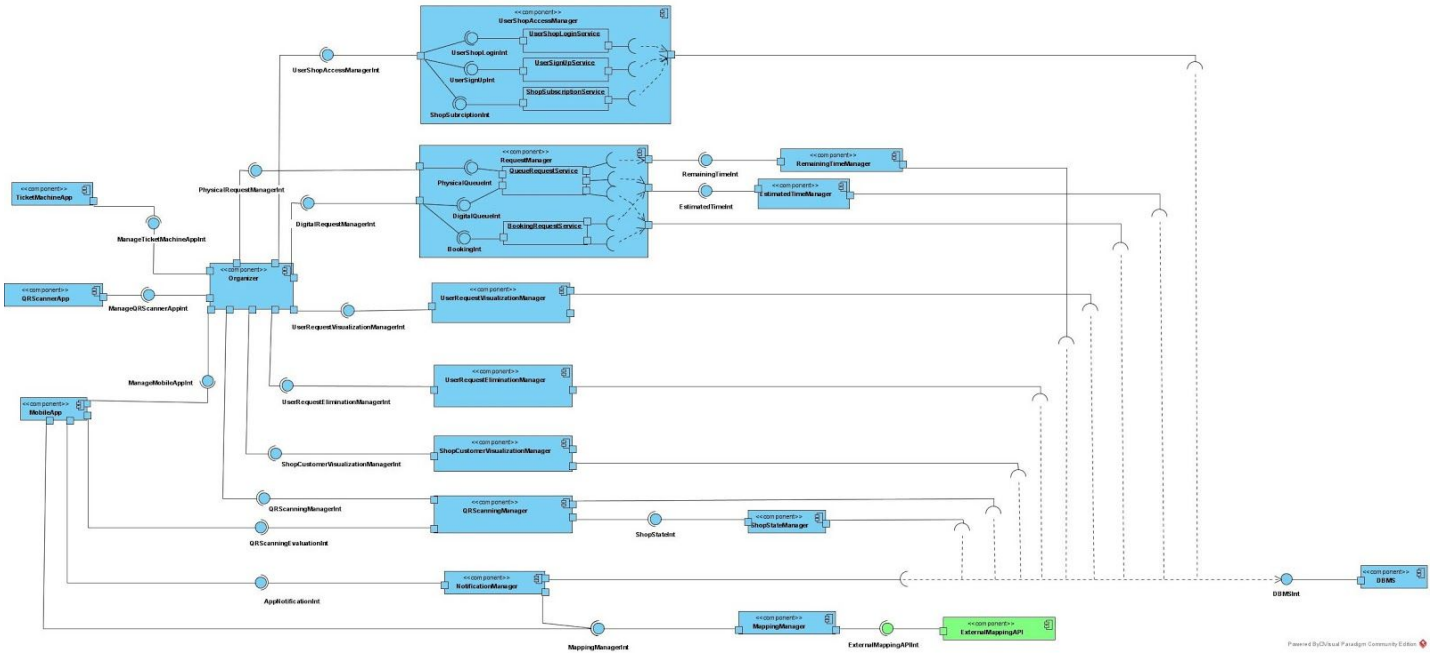
Digital users will communicate with the application server also to query information about details of their requests; the same operation can be done by the shop managers to access the customers' information and relative past and active visits: the server in the application layer communicates synchronously with the database server (data access layer) to retrieve information or asynchronously to store information when needed. The communication is initialized by the client also before entering and leaving a shop when the first scanning operation is performed.

Users can use their mobile device, as the shops, or the ticket machine to exploit CLup functionalities: the smartphone and ticket machine are linked to the application server which communicates with the database server (DBMS). Before entering the shop, the user will have to scan his QR Code: the response of the operation by the application server will be visualizable on the smartphone if he is a digital user or on a display integrated with the scanner if he is a physical user.



2.2 Component view

The following diagram contains all the components (logical and physical) of the system showing their interaction. In the diagram, only the application server is analyzed because it contains and implements the business logic of the system. The Data Access Layer and Presentation Layer are just involved in a simple way to make clearer the interaction between the different layers.



- **UserShopAccessManager**: this component manages the registration and login of shops and users. It is divided into three other components:
    - **UserShopLoginService**: this component offers the same interface to users and shops: the information required to access CLup are the same.
    - **UserSignUpService**: this component manages the user registration checking into the database the credentials inserted.
    - **ShopSubscriptionService**: this component handles the shop's subscription. It analyzes the documentation provided and the DataBase to allow the shop to exploit CLup.
- **RequestManager**: this component checks the validity of the requests considering the related shop, the opening period and other factors. It generates the object request that will be saved with the unique QRCode into the database. It is divided in:
    - **QueueRequestService**: it manages the queue request made by physical and digital users. It offers two interfaces to the different kinds of user: the information required and returned are dissimilar.
    - **BookinRequestService**: it manages the booking request made by users. It checks at the specified date, if the shop is full or not. In addition, it also manages the

"OverFlow" user request functionality taking into account all the necessary considerations.

- **RemainingTimeManager**: this component specifies the number of people in the queue before for a lined-up user and the remaining time before entering the shop. These dynamic information are continuously updated and saved in the database.

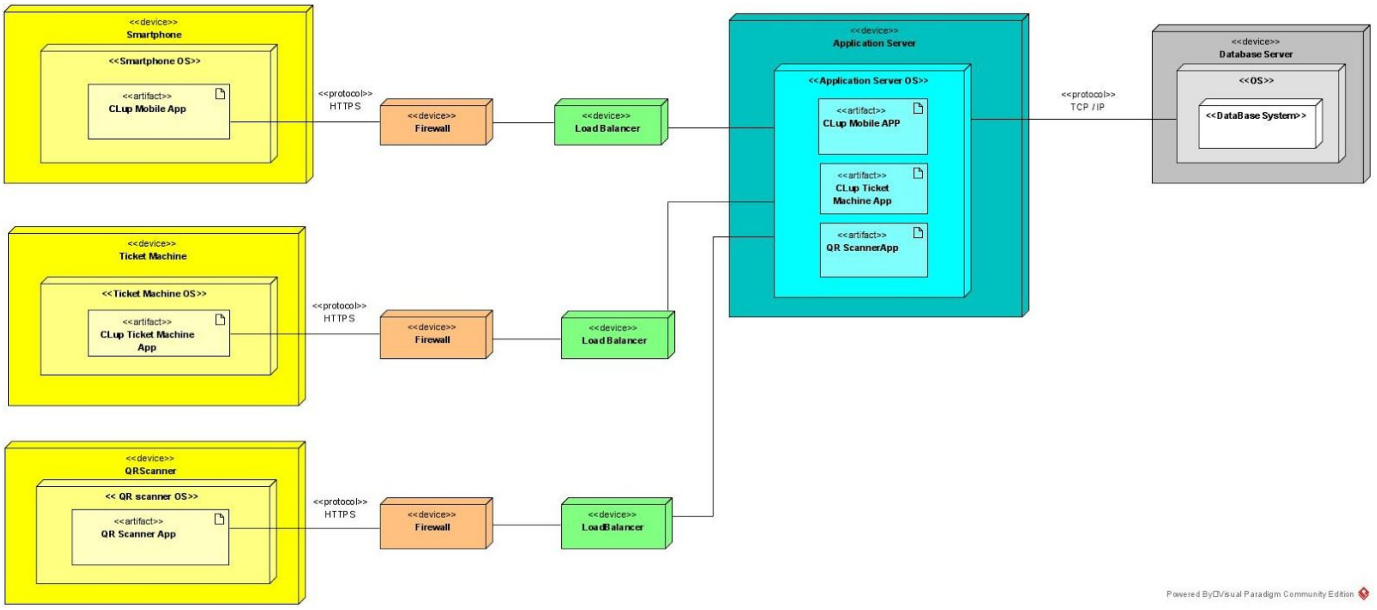
  For the physical users, this component calculates the exact time when the customer will have to approach the store: it is impossible to communicate an information as dynamic as the remaining time using a printed ticket.
- **EstimatedTimeManager**: this component manages the expected time calculation for those users who don't specificate it. This operation is done consulting the database: if the user has some past requests the system averages the effective times of those visits in the same shop, otherwise the system makes an estimation on the number of users in the queue after him and the dimension of the shop. Fewer the people in the queue, larger the time interval that will be given to the customer. Its interface is invoked by the QueueRequestService and BookingQueueService.
- **UserRequestVisualizationManager**: this component allows the digital user to visualize his active requests: for queue requests he will visualize the shop, the number of people before him, the updated remaining time, the expected time and the QR code. Instead, for booking requests, he will see the shop, the expected time, the date and hour of the visit and the QR code. The dynamic data of the queue requests are constantly updated thanks to the RemainingTimeManager.
- **UserRequestEliminationManager**: this component allows the users to cancel their active requests.
- **ShopCustomersVisualizationManager**: this component allows the shop owner to visualize the active and past customers requests related to his store.
- **QRScanningManager:** this component is involved when the user performs the scanning operation before entering and leaving the shop. It checks that the customer is allowed to enter considering the shop state and if, in that moment, it is the user's turn. Moreover, it helps to determine the effective time spent in the store. The behaviour of this component is different if the user is digital or physical: in both cases the Scanning operation is performed through the QRScannerApp component, but the result of the operation is sent to the smartphone for the digital user and shown on the display for the physical one. For this reason, MobileApp implements the interface that is used by the QRScanningManager component to forward the response. Moreover, this component manages the access of the "Overflow User".

- **ShopStateManager:** this component handles the shop state: specifically, it interacts with the QRScanning Manager to update the number of people inside of the shop in real time. It makes the DBMS aware about the beginning and conclusion of the user vitist.
- **NotificationManager**: this component alerts the user to approach the shop considering the optional position specified in the queue request and the relative remaining time. For the booked customers, it sends the notification when the visit day incomes. It is important to notice that the NotificationManager uses the interface implemented by the Mobile App: the NotificationManager calls the MobileApp methods when a notification should be sent.
- **Organizer**: this component redirects the requests and methods invoked from the shops and users to the proper component that can manage them.
- **DBMS**: this component allows all the other components in the system to interact with the database. The interface provided by this component all the useful methods to store, retrieve, update data into the database from different elements of the architecture. Every internal component uses a set of methods of its interface.
- **MappingManager**: this component acts as intermediate between the components NotificationManager and MobileApp and the ExternalMappingAPI that is an external mapping API service as GoogleMap. In this way the component NotificationManager and MobileApp will use the same interface provided by the MappingManager independently from the chosen API. The MappingManager, interacting with the external API, estimates the time necessary to reach the shop starting from the position indicated and allows the user to insert the position using the GPS in the queue request.

The system is also made up of an external component:

- **ExternalMappingAPI**: this component is the external mapping API service, such as GoogleMaps, that is useful to offer functionalities as notifications or indicating a position using the GPS.

2.3 Deployment view

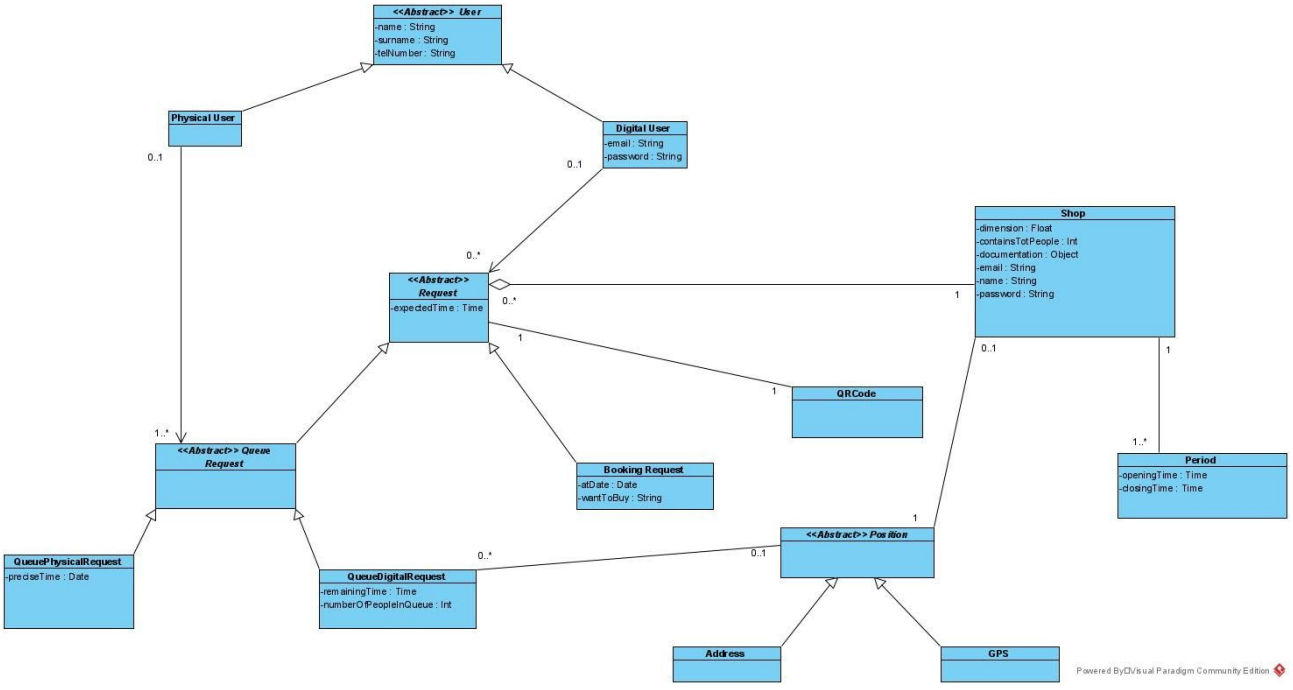


- **Smartphone**: the user and shops can use the smartphone to exploit the system functionalities. The messages are exchanged through the HTTPS protocol. The mobile application must be available for both Android and iOS to make it available on most of the devices.
- **Ticket machine**: physical user exploits it for getting tickets for a shop queue; it is connected to the application server and exchanges messages with it thanks to the HTTPS protocol.
- **QRScanner**: the scanner is used to validate users' entrance to the shop through the QR code, the integrated display shows to physical users the scanning result. HTTPS protocol makes this interaction possible.
- **Firewall**: acts as a network security device that monitors incoming and outgoing network traffic and permits or blocks data packets based on the set of security rules of the system.
- **Load balancer**: acts as a reverse proxy and distributes network or application traffic across a number of servers. It is used to increase capacity (concurrent users) and reliability of applications. In addition, it improves the overall performance of applications by decreasing the burden on servers associated with managing and maintaining application and network sessions, as well as by performing application-specific tasks.
- **Application Server**: it implements the business logic, it handles all the requests and provides the appropriate answers for all the offered services. It interacts with the mobile application, ticket machine and QR Code scanner and the integrated display.
- **Database Management System**: the data access layer is constituted by a database server that executes a relational DBMS. A Relational Database system is the most simple model, as it does not require any complex structuring or querying processes. In the Relational

Database System, there is no pattern or pathway for accessing the data, as to another type of databases can be accessed only by navigating through a tree or a hierarchical model. A Relational Database system by itself possesses qualities for leveling up, expanding for bigger lengths, as it is endowed with a flexible structure to accommodate the constantly shifting requirements. This facilitates the increasing incoming amount of data, as well as the update and deletes wherever required. This model consents to the changes made to a database configuration as well, which can be applied without difficulty devoid of crashing the data or the other parts of the database.

2.4 Runtime view

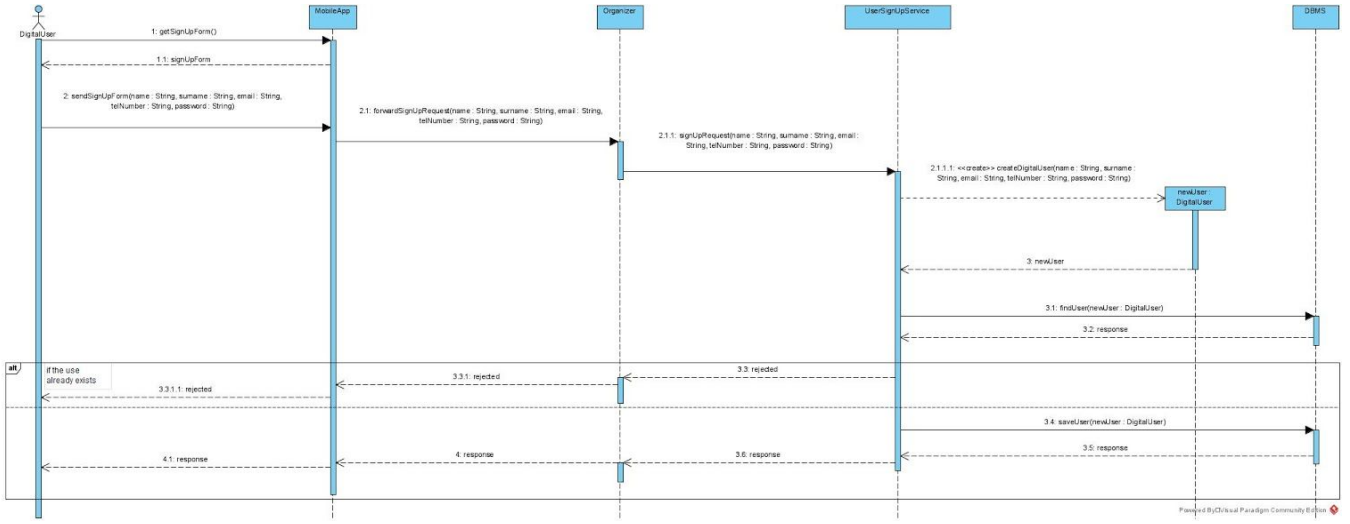
In the following, the Class Diagram for the Design Document is reported: it represents the model of the considered system.



The class QRCode has been introduced, which is linked to the abstract class Request. This class is useful to identify each request in the database and to allow the user to access the shop. In addition, the class QueueRequest is changed as an abstract class. It is implemented by the QueueDigitalRequest and QueuePhysicalRequest that are characterized by different attributes even if they offer the same functionality. A new abstract class Position is associated with none or one shop and with none or one request: it is optional to insert a position when joining a queue while is mandatory for the shop manager indicating the position of his activity. Moreover, the shop has a

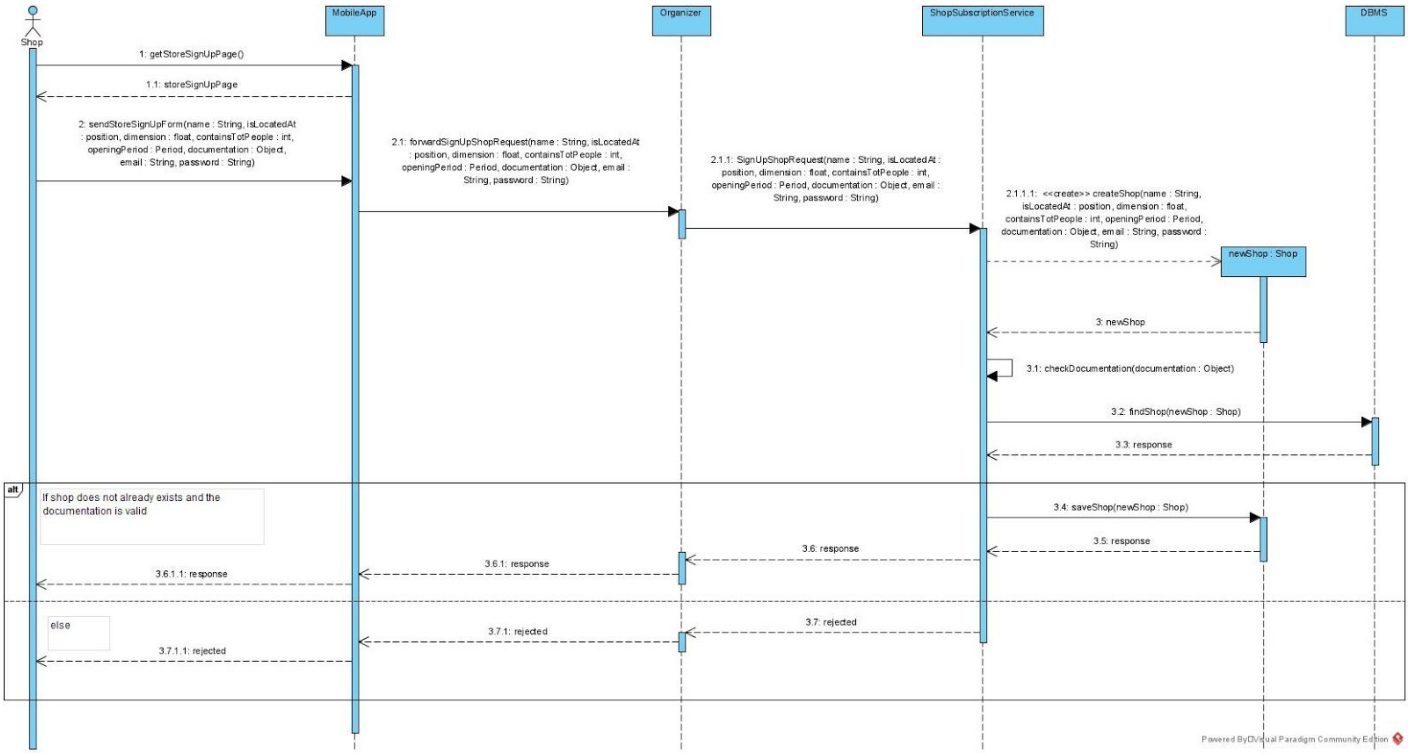
new important feature represented by the Period class: it defines the time when the shop starts working and when it ends.

2.4.1 User sign up



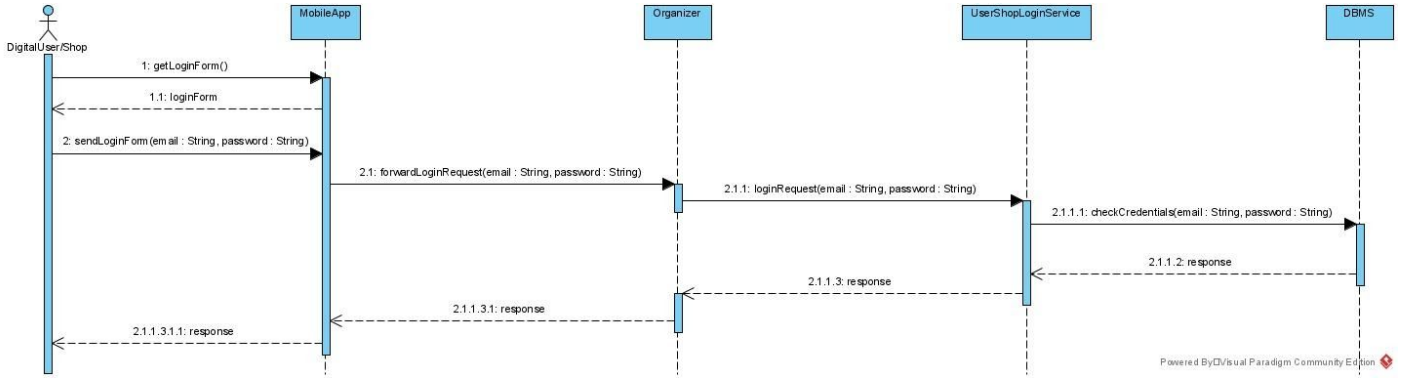
This sequence diagram represents the user registration flow: the user opens the app on his mobile phone and gets the sign up page, here he has to fill all mandatory fields that are name, surname, telephone number, email and password. This form is forwarded to the Organizer that redirect the request to the UserSignUpService component. It creates a DigitalUser object and checks, using the interface method of the DBMS, if there is already an user with the same credentials in the database. After this control, the user will receive the operation result.

2.4.2 Shop subscription



This second sequence diagram is very similar to the previous one with subtle differences related to mandatory fields: now the shop owner has to insert the name of the shop, opening periods, the number of people that can contain, its dimensions, its address, the documentation that ascertains its role, an email and a password. All these information are forwarded to the ShopSubscriptionService component and ,similarly as the previous sequence diagram, creates an object Shop. The check is made considering the documentation provided and the database.

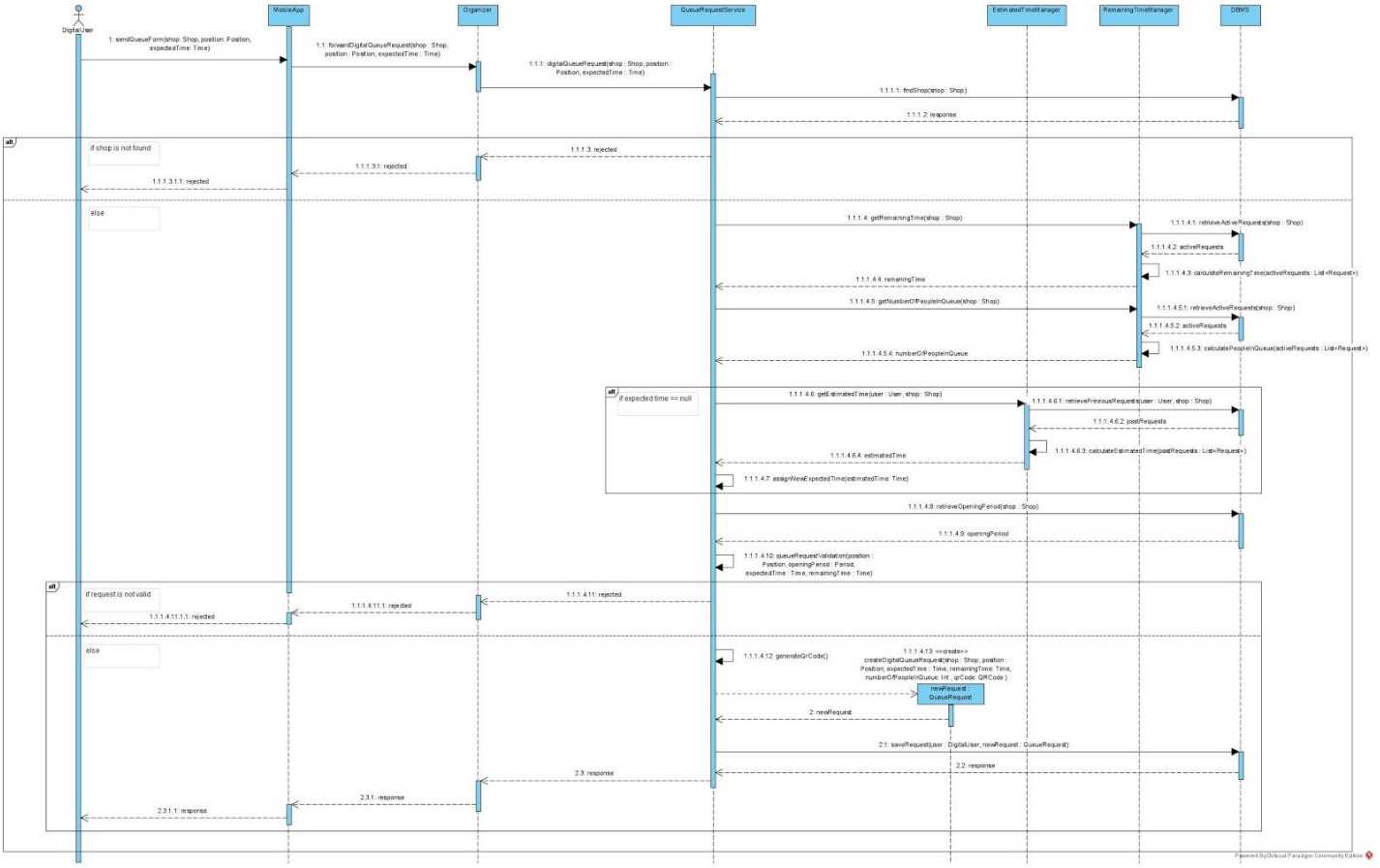
2.4.3 User/Shop login



In this sequence diagram is explained how the login operation, common to user and shop, happens. The sign up precedent operation is necessary to perform this one. The mobile app sends the compiled form with email and password to the organizer that forwards it to the

UserShopLoginService component. It checks if the specific user or shop exists in the database and if the relative credentials are correct. After receiving a response about the operation result, this one is sent back to the user/shop.If the result is positive the login is successful, otherwise it should be repeated.

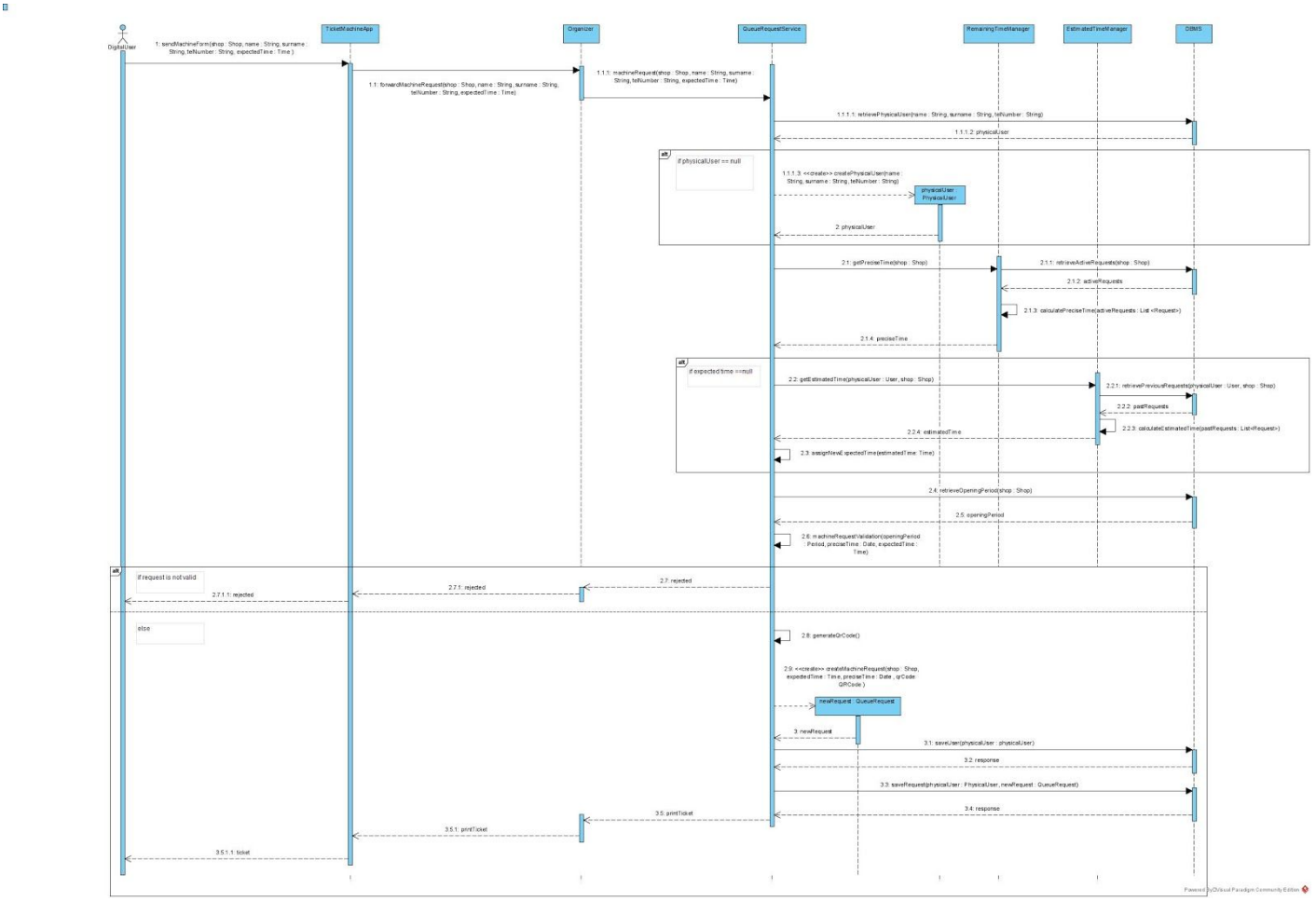
2.4.4 Joining a queue



This sequence diagram explains the queue join operation made by a digital user. This diagram shows how the system reacts to the queue request receipt thanks to the Organizer, who receives the message from the MobileApp and forwards it to the QueueRequestService: it checks if the inserted shop exists with the help of the DBMS, it uses the RemainingTimeManager interface to get the actual remaining time and the current number of people that will precede him in the queue. The RemainingTimeManager retrieves this two information querying the DBMS: specifically it invokes a method to get the active requests (booking and queue) of that shop. In this way, it can calculate the remaining time and the people before the user taking into account the real time queue of that shop. Then, if the user has not inserted any expected period, the QueueRequestService involves the EstimatedTimeManager: this component retrieves the past requests of the user in that specific shop in order to estimate a possible time that will be spent by the user in that shop. This value estimated

will be assigned as the expected time. Lastly, some checks must be done to control the remaining time, expected time and the relative opening period of the specified shop. After that, the QueueRequestService generates a QR code and creates a Request object that will be associated to the user and saved in the database.

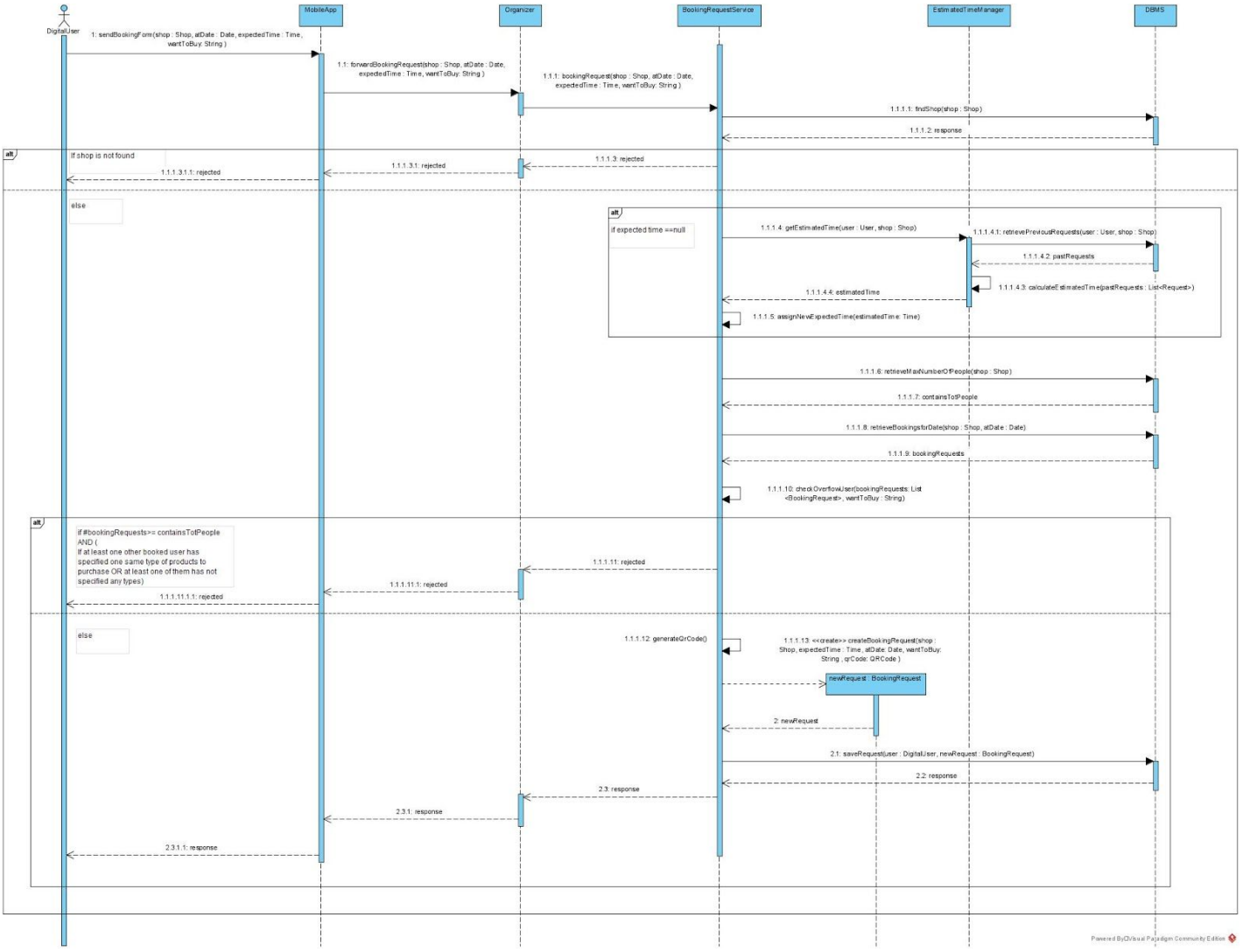
2.4.5 Getting a ticket



This sequence diagram explains how a physical user gets a ticket to join a shop queue. After approaching the ticket machine outside the chosen shop, the user asks for the form that he has to fill with his name, surname, telephone number and the expected time. The TicketMachineApp component sends the form to the Organizer, that forwards it to the queue request service; it searches the user in the DBMS and creates a new one if it doesn't exist yet. Differently from the digital case, the RemainingTimeManager does not calculate the dynamic countdown before getting in the shop but gives to the physical user a precise time to access the store: this operation is performed taking into account the active requests of that grocery store. If the user didn't specify the expected time, the EstimatedTimeManager component gets the estimated time considering the user's previous visits. It is possible that this physical user is also a registered (digital) user: in this case the previous

digital requests are analyzed. Lastly, a check on the hour related to the shop opening period is done and, if this latter is successful, a QR code is generated and a new request is created, associated to the user and saved in the DBMS. The operation finishes with the QueueRequestManager response to the TicketMachineApp, that can order to print the ticket with all information or an error message.

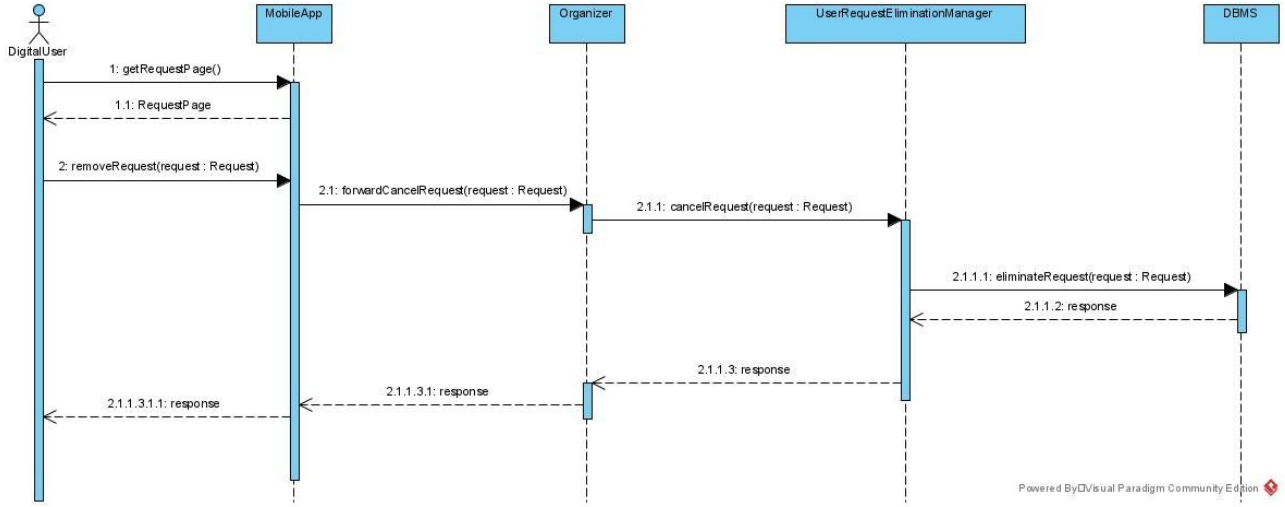
2.4.6 Booking a visit



This function's sequence diagram is related to digital users and allows them to choose a day and an hour to enter a grocery shop without waiting in the virtual queue. After the user login on the app he asks for the booking page where he has to fill the mandatory fields: shop, day and hour, and he could specify the expected time and types of products that he has to purchase. The mobileApp sends the form to the Organizer that forwards it to the BookingRequestService component that manages it. After the first control on the specified shop existence, the system checks if the form contains an expected time and, if does not, the estimatedTimeManager returns it considering previous visits.

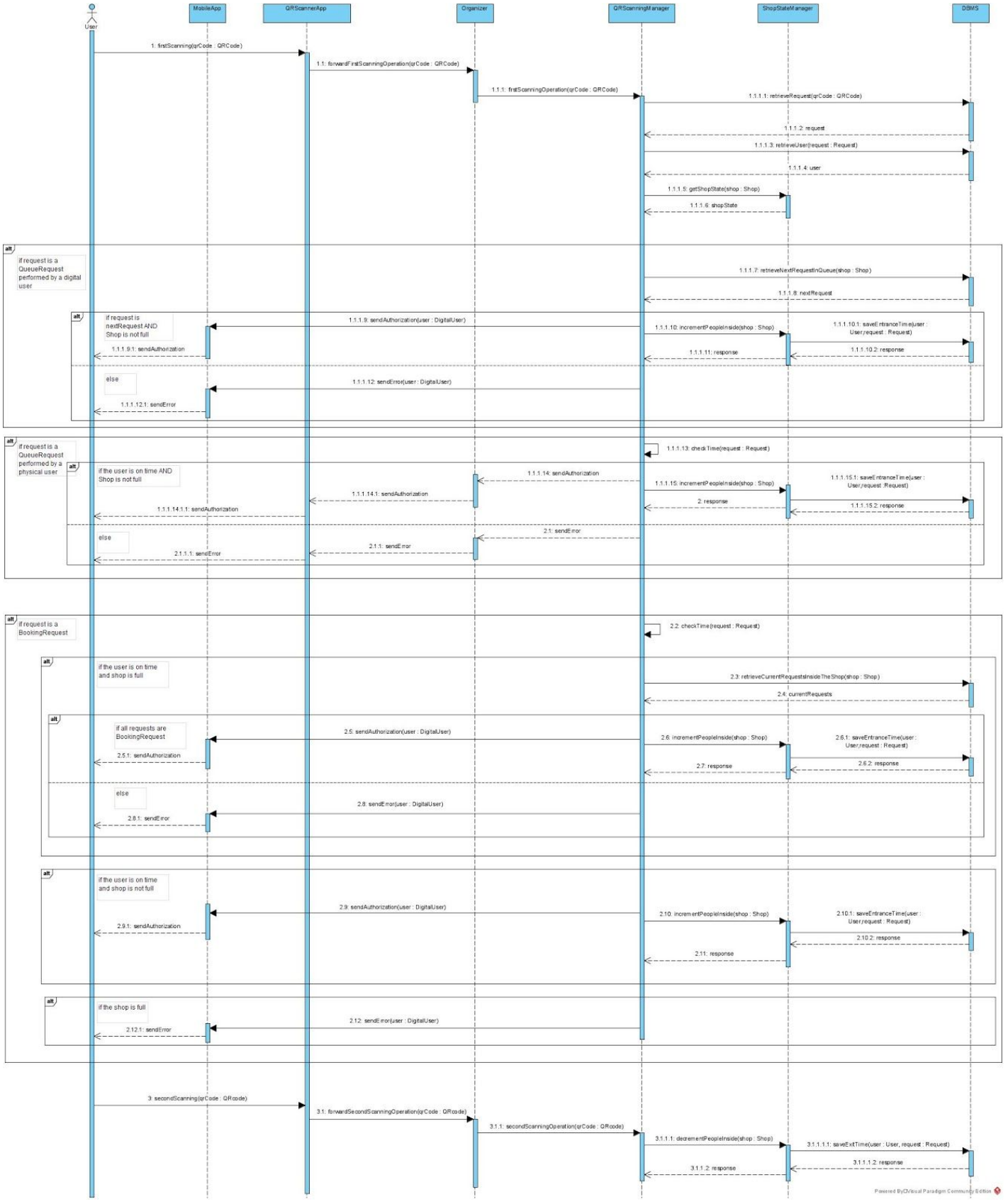
Then, the BookingRequestService retrieves the maximum number of simultaneous people inside the shop and the booking requests that are expected to be in the shop in that time period when the actual user books his visit. Before performing all the necessary considerations, the component examines, if exists, the kinds of product specified by the other booked user in the same time slots and the optional kind of the actual request. Here there are some conditions : if the shop is full of booked users for that time slot and there is at least one other request with at least one common kind of product or one of them (including the actual request) has not specified any kinds of product , the request is rejected. Else, if the shop is not full or it is full and all kinds of products are different between and specified by all the contemporary booked users inside, the user is allowed to book that visit. In these cases the QR code is generated, the new booking request is created and saved in the DBMS and the response is sent back to the user passing for the organizer and the mobileApp.

2.4.7 Canceling a request



The user gets the request page using the MobileAppInterface and chooses the request he wants to cancel. The cancelingRequest is sent to the Organizer that dispatches it to the UserRequestEliminationManager. This component eliminates the request from the database.

2.4.8 Entering and leaving the shop



In this sequence diagram the user's QR scanning operation after approaching the shop is represented. The user interacts with the QRScannerApp component that forwards the QR code to

the organizer. It sends the code to the QRScanningManager that manages all checks and operations. Using the QR code, the manager retrieves from the DBMS the related request and from the request, it retrieves the related user. Then the QRScanningManager component asks the ShopStateManager the shop state.

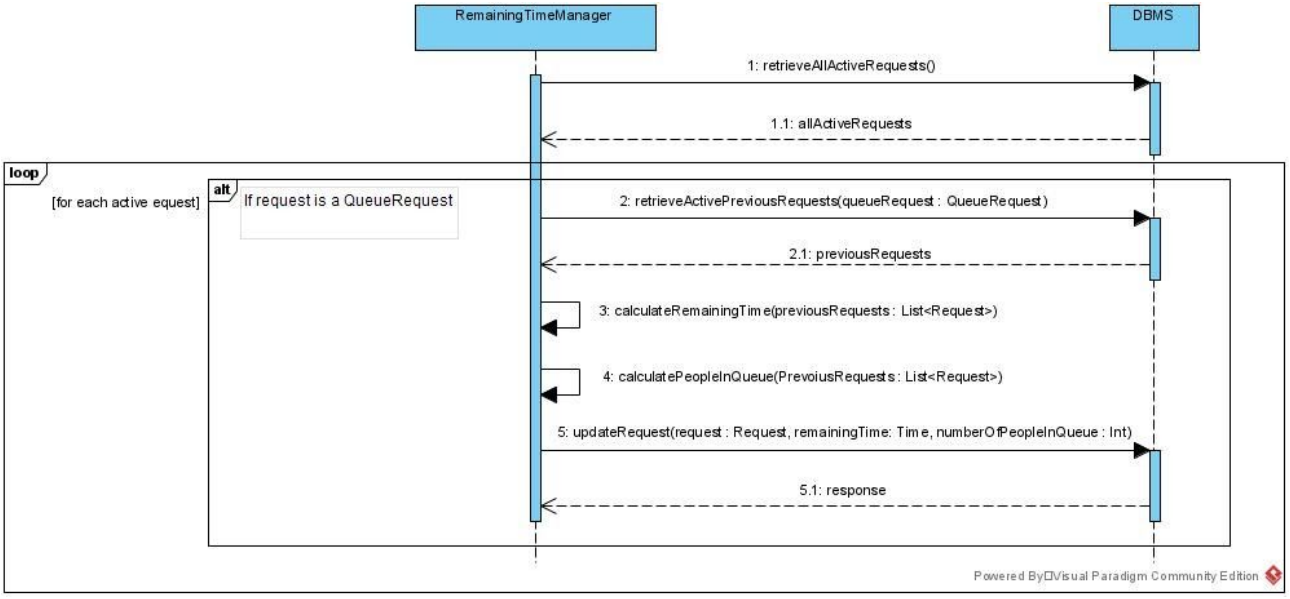
The operation is divided in three cases:

1. the request is a QueueRequest performed by a digital user: in this case the QRScanningManager asks the DBMS the queue request with the minimum remaining time (the next one visit). If the scanned request is the one with the minimum remaining time and the shop is not full, the user is allowed to enter and the system sends an authorization message to the MobileApp. Otherwise, the system sends an error message;
2. the request is a QueueRequest performed by a physical user: in this case the system checks the current time related to the request's precise time: if the user is on time and the shop is not full the customer is allowed to enter and the system sends an authorization message to the QRScannerApp that displays it on the integrated display. Else, on the display is showed an error message;
3. the request is a BookingRequest: in this last case the system checks the current time related to the request's date and hour: if the user is not on time the scanning operation is refused and an error message is sent to the mobile app to that user; else, if the shop is not full the user is allowed to enter with an authorization message. Nevertheless, there is a specific condition for the overflow user: if the shop is full the message authorizes him if and only if all the users inside the shop are booked users, else the message contains an error.

When a user enters the shop the QRScanningManager sends the event to the ShopStateManager that increments the number of people inside that shop and tells the DBMS to save the user's entrance time.

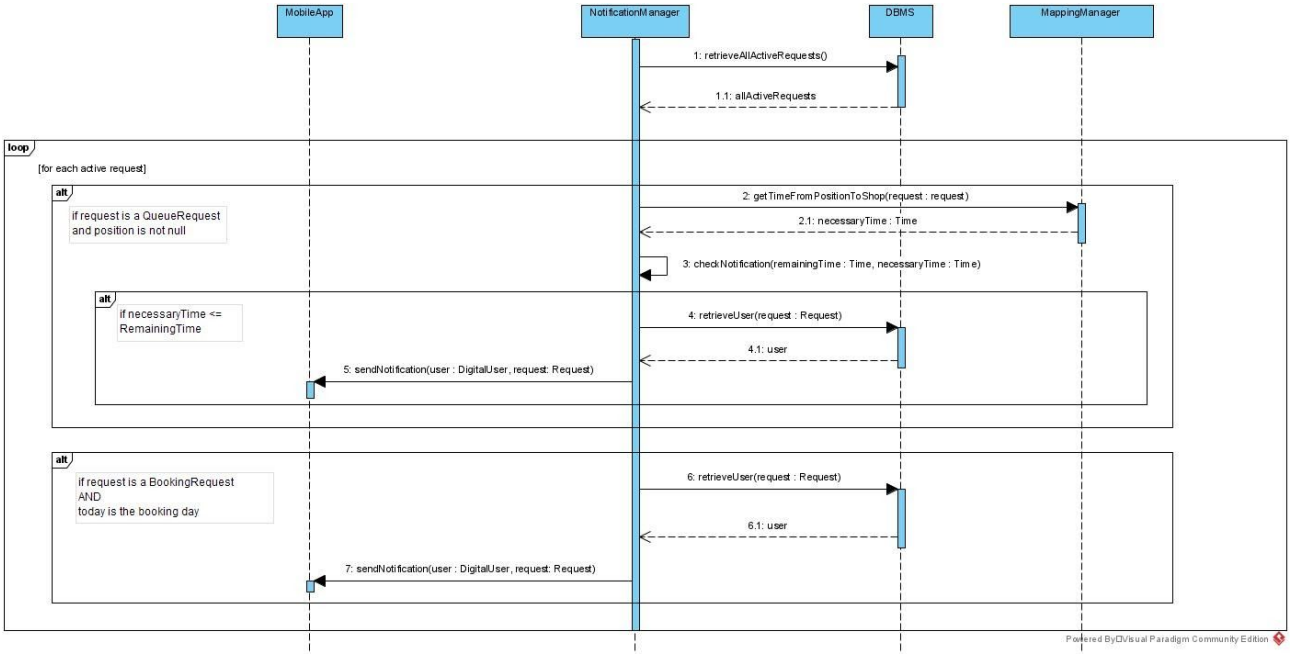
When a user scans the QR code for a second time the QRScannerApp and the Organizer forward the QR code to the QRScanningManager. It also involves theShopStateManager that decrements the number of people inside that shop and sends to the DBMS the leaving time. It allows the system to determine the effective time spent in the shop and make it aware about the completed request.

2.4.9 Update Remaining Time



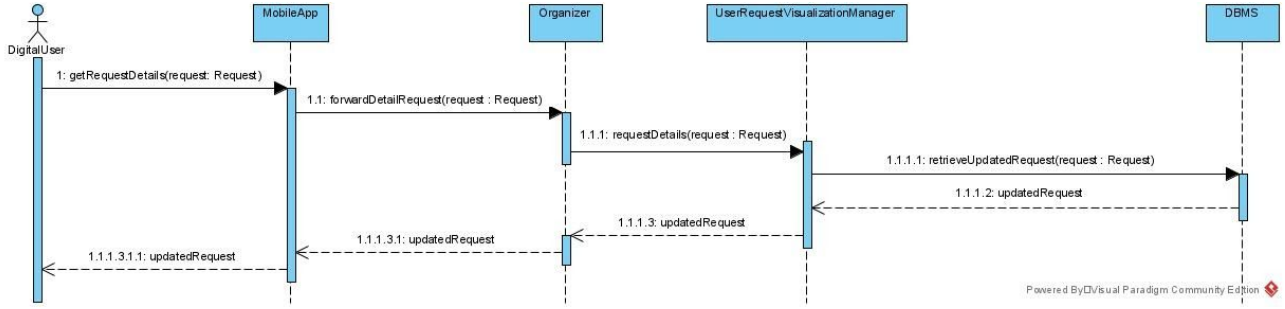
This sequence diagram shows a procedure that updates the remaining time and the number of people in the queue for arch active requests. This operation starts every 2 minutes and involves only the RemainingTimeManager and the DBMS. The first component retrieves all the active requests for each shop and then it loops on them: if he identifies QueueRequest, he takes all the active previous requests that represent the customers lined up before him in the queue. In this way, the RemainingTimeManager updates the remaining time and the number of people before that specific request. He sends to the DBMS the changes performed.

2.4.10 Notification



This sequence diagram explains how and when a notification is sent to a user. This operation is done every five minutes and, after recovering all active requests, the NotificationManager controls, for each of them, if it is a queue or a booking request. For queue requests where position is not null the manager asks the MappingManager to calculate the necessary time to reach the shop from the specified position and, if the necessary time is less or equal to the remaining time, it retrieves the user and sends a notification to him. For booking requests, the NotificationManager checks if the booking date is today, if yes it retrieves the user and sends a notification to him.

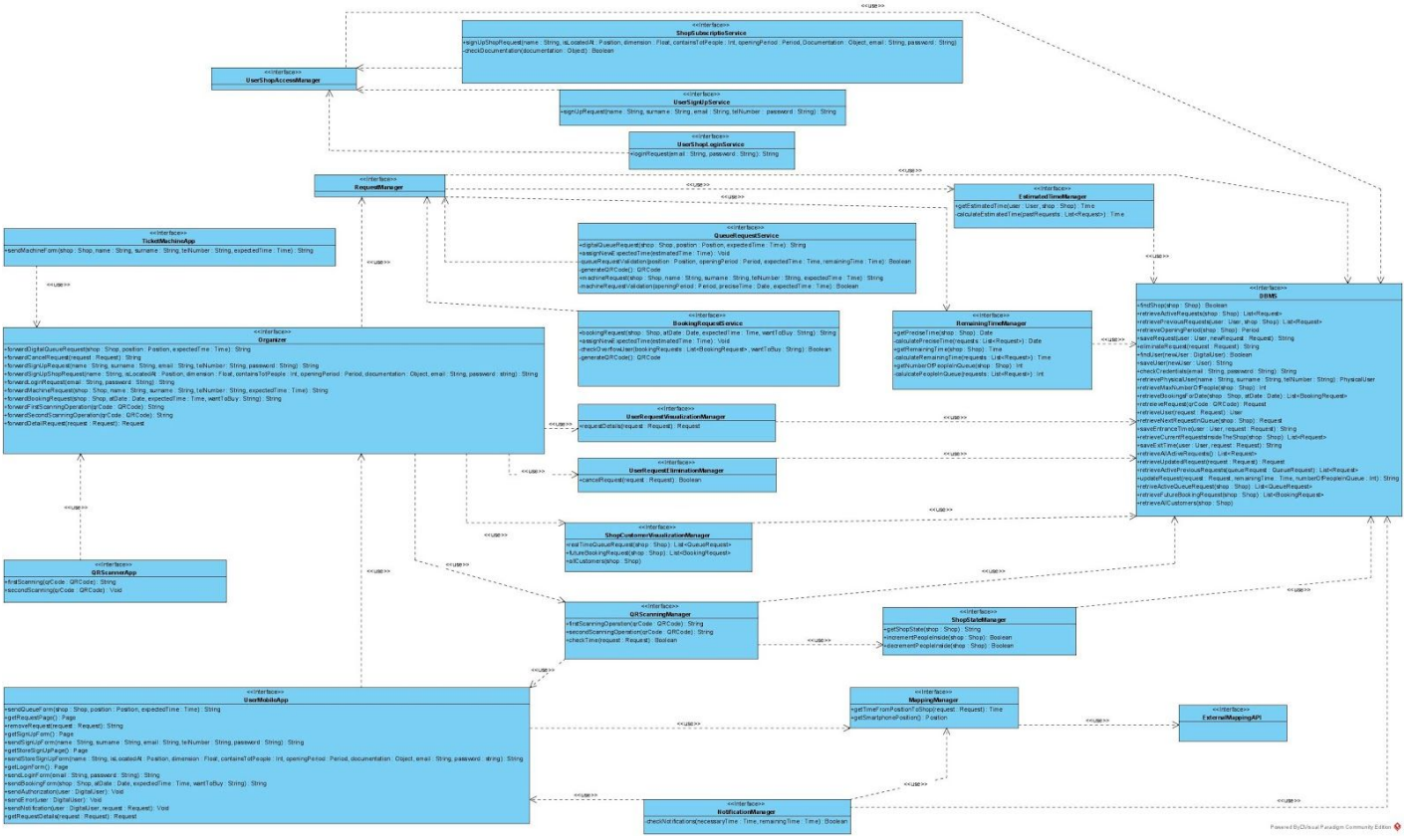
2.4.11 User Request Visualization



The system allows the user to visualize the details of his request. The customer, using his smartphone, chooses the active request and, starting from the MobileApp, it is sent to the UserRequestVisualizationManager through the Organizer. This component simply asks the DBMS the latest updated version of this request and the new information is sent back to the user.

2.5 Component interfaces

In the next diagram are described the main methods which can be invoked on the interfaces and their interactions, referring to the most important processes reported in the runtime view section.
It is important to notice that the methods written in the Component Interfaces diagram are not to be intended exactly as the methods that the developers will write, but they are a logical representation of what component interfaces have to offer. They will be adapted facing the various aspects that will come out during the implementation of the code.



2.6 Selected architectural styles and patterns

2.6.1 Client - Server and communication protocol

The paradigm used is the Client - Server one; the paradigm divides software into these two categories. A client initiates a connection and sends requests, whereas a server is software that listens for connections and processes requests. Nevertheless, in the system this condition is not ever respected: generally is the MobileApp, using the Organizer, to ask the Application Server for a service but when the system needs to send a notification, the NotificationManager performs a call on the Mobile App that changes its role and becomes a server. The same reasoning can be done considering the QRScanningManager.

The communication protocol used to exchange messages is HTTPS. Using TSL when dealing with sensitive data in order to guarantee the security and the reliability of the connection. Regarding the format of the transmitted data, JSON is used for data exchange between client and server.

The system uses cache on the client side in order to reduce the number of communication between clients and servers, speeding up the communication.

2.6.2 Three Tier Architecture

As we have already mentioned, we choose to use a multitiered architecture because it reduces the complexity of the system for the following reasons.
Each tier can run on a separate operating system and server platform - e.g., web server, application server, database server - that best fits its functional requirements. Each tier runs on at least one dedicated server hardware or virtual server, so the services of each tier can be customized and optimized without impacting the other tiers.
This architecture allows a faster development because each tier can be developed simultaneously by different teams, an organization can bring the application to market faster, and programmers can use the latest and best languages and tools for each tier. An improved scalability and reliability are offered: any tier can be scaled independently of the others as needed and an outage in one tier is less likely to impact the availability or performance of the other tiers.
Moreover, an improved security is guaranteed: the presentation tier and data tier can't communicate directly, a well-designed application tier can function as a sort of internal firewall, preventing SQL injections and other malicious exploits.

2.6.3 Model-View-Controller

The Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Each architecture component is built to handle specific development aspects of an application.
A view is that part of the application that represents the presentation of data. The Controller is that part of the application that handles the user interaction. The model component stores data and its related logic. It represents data that is being transferred between controller components or any other related business logic. We choose to use this pattern in order to guarantee an easy code maintenance

and a simpler way to extend it. The MVC allows also to test and develop the component independently and reduces the complexity by dividing the application into three units.
The client uses the front-end and so the view to interact with the controller that handles the information received from and sent to the database (the model).

Chapter 3

User interface design

The following pictures represent the user and shop interfaces of the CLup application. In the first picture we can find the CLup icon located in a smartphone homepage (1). Opening the application, clients and shop owners will visualize the login and signup page (2). If the shop or client is not already registered, they should compile the subscription form (3,4), reachable by clicking on relative buttons from the first page. After this operation, they can log in to CLup indicating email and password in the appropriate form (5). The shop's and user's homepage are different: the customer can immediately visualize two buttons in order to join a queue or to book a visit (6). Joining a queue requires the user to insert the interested shop, a position and the visit's expected time (7) while the booking form needs mandatorily shop and date and hour, optionally kinds of products to purchase and the visit's expected time (8). Additionally to these operations, the user is able to open a side menu (9) where he can choose to return to the homepage, to see his active requests or to log out CLup. If he chooses the request option, he will see the list of his active requests (10), and clicking on one of them, he accesses the request's details page where he can read the relative information and the QR code (11).

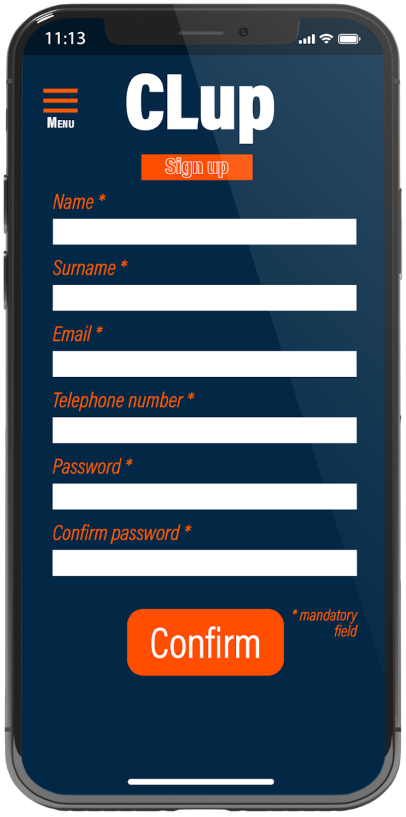
On the other hand, the shop owner's homepage contains three buttons that allow him to see the active queue for his shop, the future booked clients and all the CLup past clients that have been in the store (12). In the 13th figure we can see an example of the past clients list: each customer name, surname, telephone number and email is visualizable (13). The shop owner, in the side menu, will see the logout option when he's on the homepage (14) and also the homepage option when he's somewhere else.

When the physical user approaches the ticket machine, he will visualize on its screen a form where he can insert the necessary data to become a lined-up customer (15). Whether this operation is successful, the machine will print the ticket with all visit's information (16).

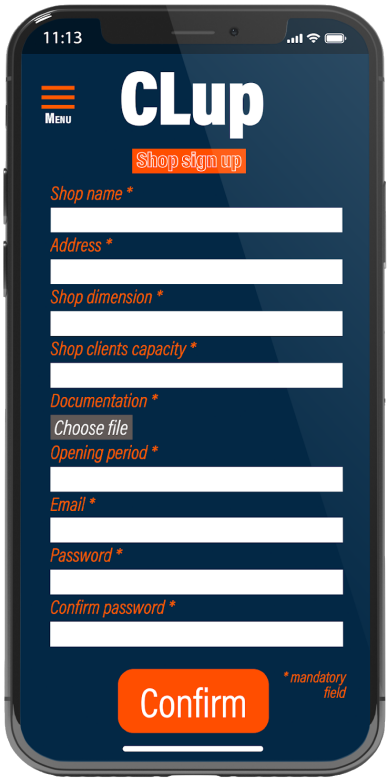
As we already said, the display will be involved only when the scanning operation is performed by a physical user otherwise the notification of that operation will be sent on the digital user mobile device. Consequently, the messages layouts are the same and the only difference is the device where the notifications will be sent. In the last four mock-ups we designed the scanner's integrated display messages that are the same for the smartphone. There are four possible cases: the display communicates availability for scanning a QRCode (17), the user scans his QRCode and he is allowed to enter (18), it is not his turn (19) or the shop is full (20).

1. CLup icon

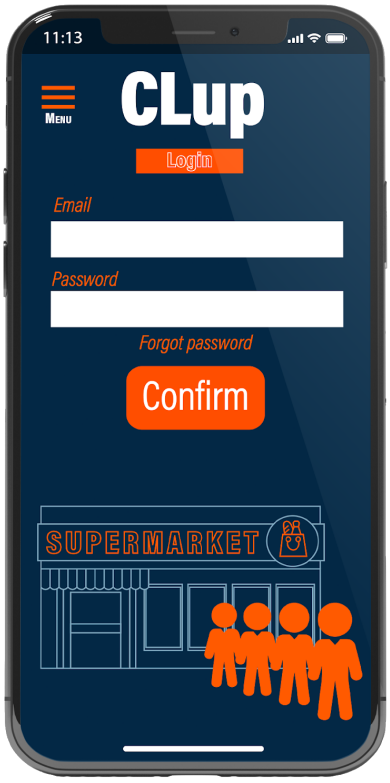
2. CLup homepage without loggin in



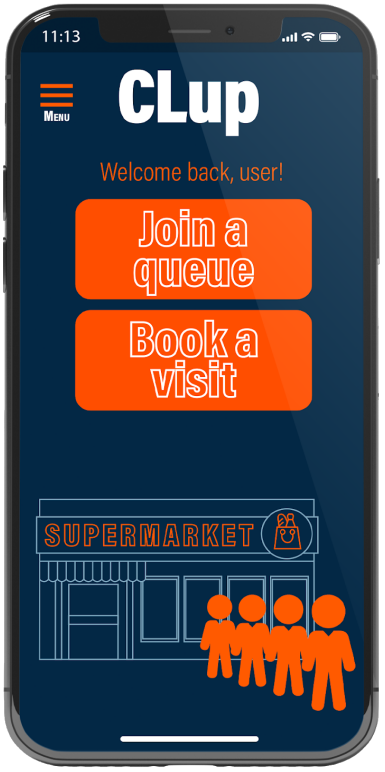
3. User sign up page



4. Shop subscription page



5. CLup login page

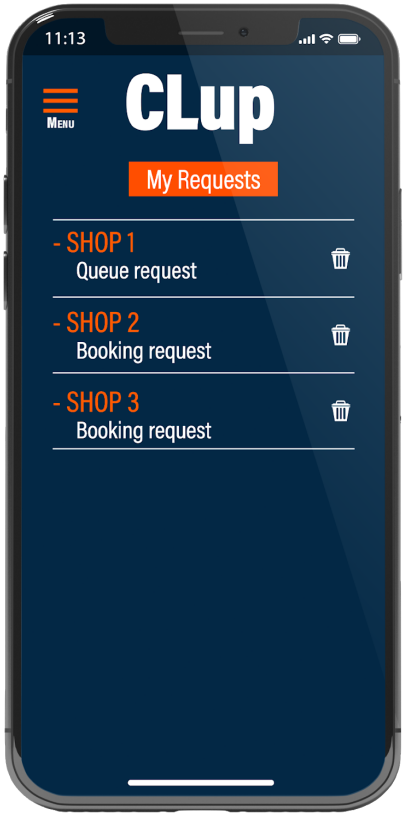


6. User's homepage

7. Joining a queue form

8. Booking a visit form

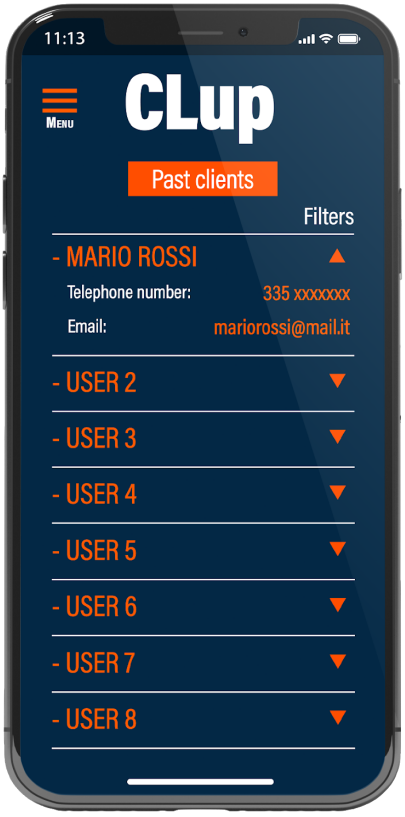
9. User menu



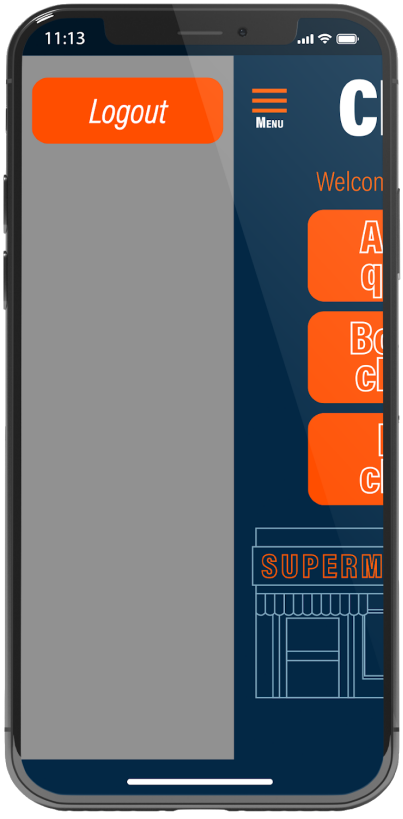
10. User's requests page

11. Request's details page

12. Shop's homepage



13. Past clients page



14. Shop menu

15. Ticket machine queue form

16. Printed ticket

QR scanner display mock-ups

17. Scanner waiting for QR Code

18. Authorization after QR scanning

19. Wrong turn error message

20. Full shop error message

Chapter 4

Requirements traceability

R1	The system shall allow the user to register for the application and log in it.
R2	The system shall allow the user to send a request for joining a queue.
R3	The system shall allow the user to choose the shop that he wants to visit.
R4	The system shall allow the user to join a queue using a ticket machine.
R5	The system shall allow the digital user who joins a queue to optionally communicate his position using GPS or specifying an address.
R6	The system shall allow the user to optionally communicate an expected duration of the visit inside the shop.
R7	If the user does not insert any expected time spent into the shop, the system calculates it analyzing statistics.
R8	The system stores the data about users and their active and past requests.
R9	The system provides a unique QR code for the specific user and request.
R10	The system calculates dynamically the remaining time for each digital user in the queue using the expected time specified or estimated by CLup.
R11	The system shall allow the user to check the state and the details of his queue active requests.
R12	The system shall allow the user to check the state and the details of his booking active requests, in particular the shop and the booking date and hour.
R13	The system shall allow the digital user to cancel the required service.
R14	The system shall allow the user to book a visit to a grocery shop indicating the specific date and hour.
R15	The system shall allow the booked user to optionally specify the kind of products he wants to purchase.
R16	The system shall allow a user to book a reservation if, for the chosen date and hour, the shop is full yet but, specifying types of product, the user has

	to visit different departments than all the other customers. This user is considered as an “Overflow User”.
R17	The system shall allow the shop owner to register their shop to the application and log in it.
R18	The system shall allow the shop to communicate the necessary documentation, store dimensions, maximum number of people inside of it and other information.
R19	The system stores the data about the registered shops.
R20	The system shall allow the shop to visualize the related real time queue, booked clients and the past requests.
R21	If the user scans the QR code for entering when the shop is full but it is his turn the system allows him to enter if and only if he’s an “overflow” user, otherwise the system rejects the scanning operation and shows an error message.
R22	If the user scans the QR code for entering when it is not his turn, the system does not allow the user to enter by rejecting the scanning operation and showing an error message.
R23	If the user scans the QR code for entering when the shop is not full and it’s his turn, the system allows the user to enter it confirming the scanning operation.
R24	The system sends notifications to make the digital user aware of the incoming shop visit.

Acronyms:

MobileApp: MA

TicketMachineApp: TMA

QRScannerApp: QSA

UserShopAccessManager: USAM

- UserShopLoginService: USLS
- UserSignUpService: USUP
- ShopSubscriptionService: SSS

RequestManager: RM

- QueueRequestService: QRS
- BookingRequestService: BRS

RemainingTimeManager: RTM

EstimatedTimeManager: ETM

UserRequestVisualizationManager: URVM

ShopCustomerVisualizationManager: SCVM

UserRequestEliminationManager: USEM

QRScanningManager: QSM

ShopStateManager: SSM

NotificationManager: NM

Organizer: O

DBMS

MappingManager: MM

ExternalMappingAPI: EM

	MA	TMA	QSA	USLS	USUP	SSS	QRS	BRS	RTM	ETM	URVM	USEM	SCVM	QSM	SSM	NM	MM	EM
R1	**X**			**X**	**X**													
R2	**X**						**X**											
R3	**X**						**X**	**X**										
R4		**X**					**X**											
R5	**X**																**X**	**X**
R6	**X**	**X**					**X**	**X**										
R7										**X**								
R8							**X**	**X**										
R9							**X**	**X**										
R10									**X**									
R11	**X**										**X**							
R12	**X**										**X**							

R13	**X**											**X**						
R14	**X**							**X**										
R15	**X**							**X**										
R16	**X**							**X**										
R17	**X**			**X**		**X**												
R18	**X**					**X**												
R19						**X**												
R20	**X**												**X**					
R21	**X**		**X**											**X**	**X**			
R22	**X**		**X**											**X**				
R23	**X**		**X**											**X**	**X**			
R24	**X**															**X**	**X**	

As it can be seen through this matrix, every component covers at least one requirement and every requirement is covered by at least one component. There are no unnecessary components.

The DBMS is not indicated in the matrix because it represents the Data Layer in the Three Tier Architecture and every requirement requires data to be fulfilled.

In the traceability matrix , the **Organizer** component is not mentioned for the sake of simplicity, but they are directly or indirectly connected to the fulfillment of the system functionalities because it routes to the right component every message coming from the client's side.

Chapter 5

Implementation, integration and test plan

5.1 Implementation Plan

The subsystem will be implemented, but also testing and integrated exploiting a bottom- up strategy considering the relevance of each component in relation to the entire system.

The bottom-up approach is an implementation strategy in which the modules at the lower level are implemented with the higher modules until all the modules and aspects of the software are completed properly. Additionally, in bottom-up testing, components at lower hierarchy are also tested individually after being implemented and then the components that rely upon these components are tested. Though top level components are most important, yet they are tested last using this strategy of integration testing.

Due to the structure of the system, there are components that rely on others, so it is necessary to identify the relation between them to understand the priority order.

1. It is clearly visible that the **DBMS** should be implemented as first for two main reasons: its interface is used in every system functionality and is independent from all the other components.
2. After we can starts the implementation of many potential components UserShopAccessManager, RemainingTimeManager, EstimatedTimeManager, UserRequestVisualizationManager, UserRequestEliminationManager, ShopCustomerVisualizationManager, ShopStateManager and MappingManager. The latter rely on the External MappingAPI but it is already implemented by a trusted provider. The others not mentioned rely on some of these components, so they can't be implemented. The **UserShopAccessManager** is probably the most relevant component among them because it allows the registration and login of shops and users in order to exploit the CLup functionalities.
3. Then, we can proceed to implement the **RemainingTimeManager** and **EstimatedTimeManager** whose interfaces are fundamental to allow the RequestManager to perform his duty properly.
4. Due to its importance, the **RequestManager** can be now implemented.

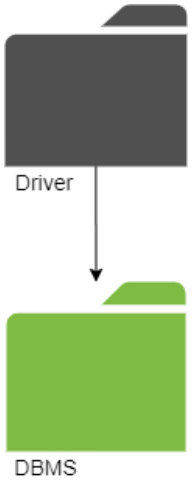
5. After that, it is possible to implement the **UserRequestVisualizationManager**, **UserRequestEliminationManager** and **UserCustomerVistaulizationManager** that allow users and shops to manage their relative requests.
6. Then, the component **ShopState** and **MappingManager** can be completed. The latter use an interface of an external component to offer his service.
7. After the implementation of these lower level componentes, **NotificationManager** and **QRScanningManager** rely on the implemented components and so they can join the system.

The **Organizer** is implemented at the end: it redirects the methods to the right component and acts a crucial role in the system architecture. The **MobileApp**, **TicketMachineApp** and **QRScannerApp** are implemented simultaneously with the operation described. As soon as one or more components are implemented,unit-tested and integrated, the testing is performed on the new structure created in order to find possible bugs and problems.

The **ExternalMappingAPI** is not considered in the implementation and unit - testing phase because it is an external service provided by a trusted company.

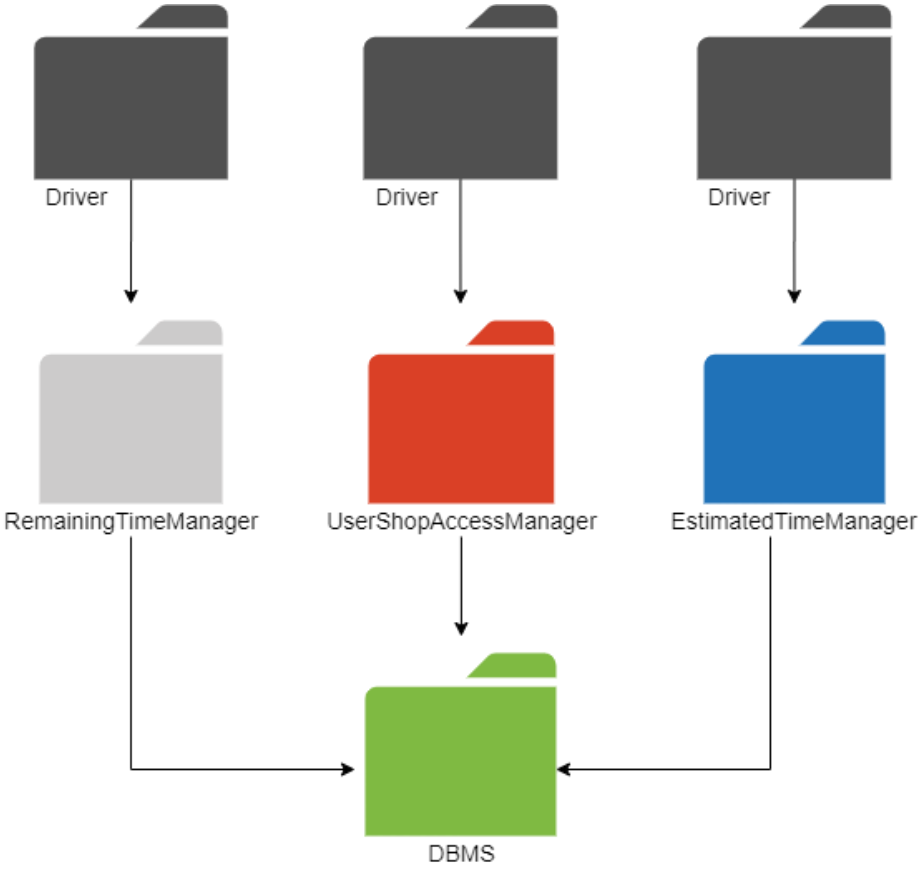
5.2 Integration strategy

1. At first, the DBMS is implemented and is associated with a driver that substitutes the component still under implementation. In this way, the DBMS component can be unit tested.

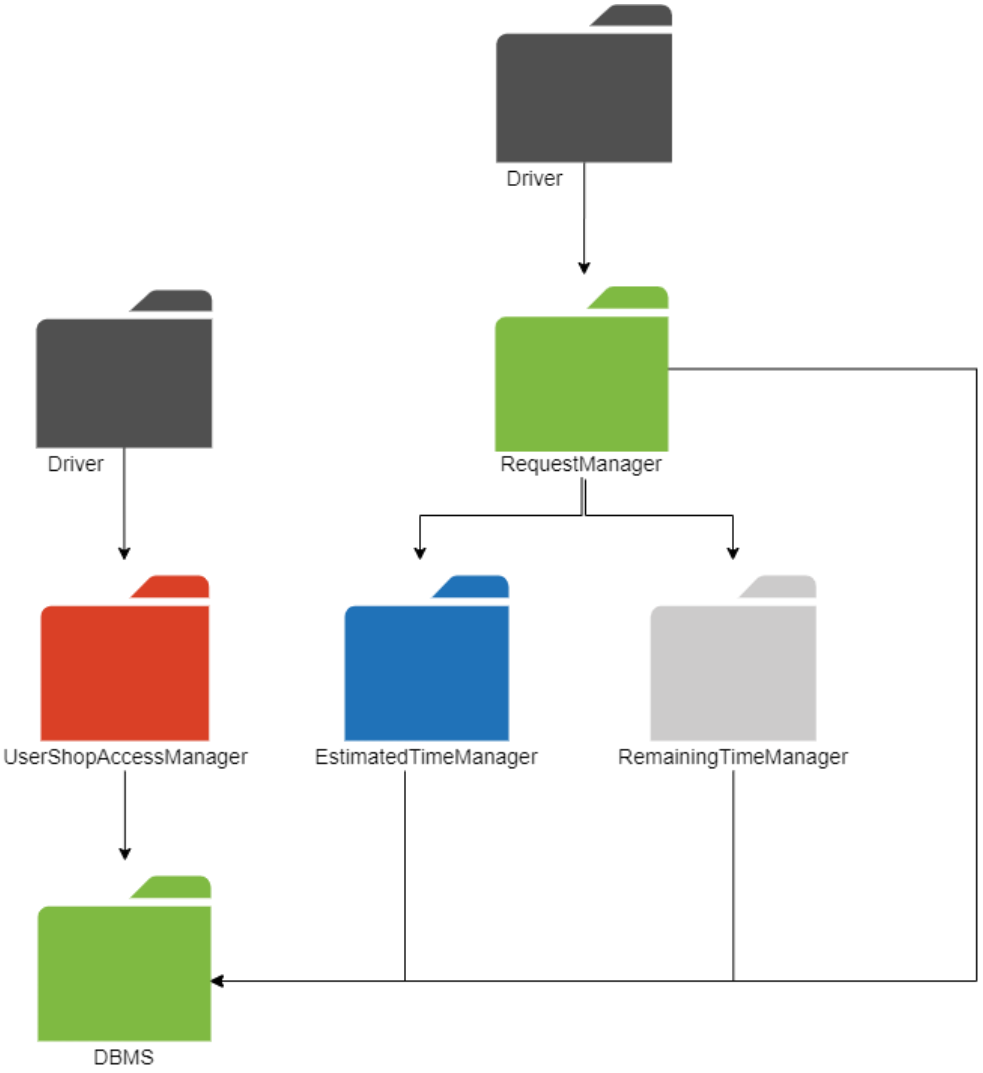


2. Then to implement and test the functionality of login e subscription by users and shops, the UserShopAccessManager is needed.

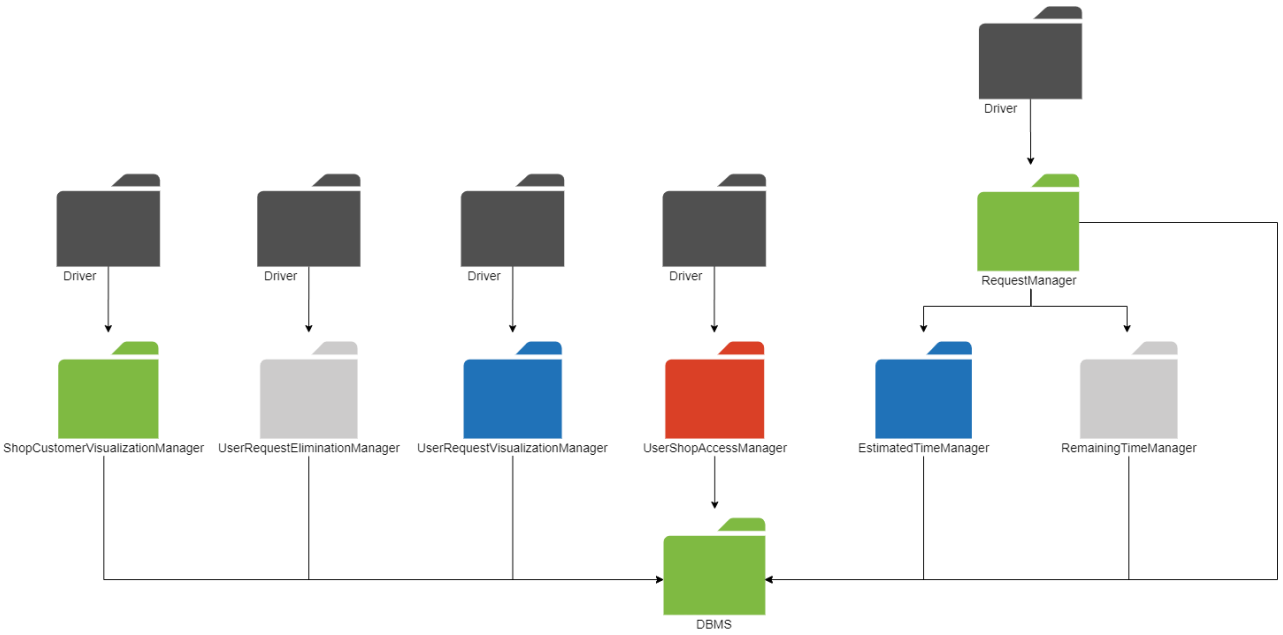
3. Then the implementation of EstimatedTimeManager and RemainingTimeManager can be done in parallel because there is no dependency between them. They have to be integrated and tested in the system before the RequestManager that relies on them.



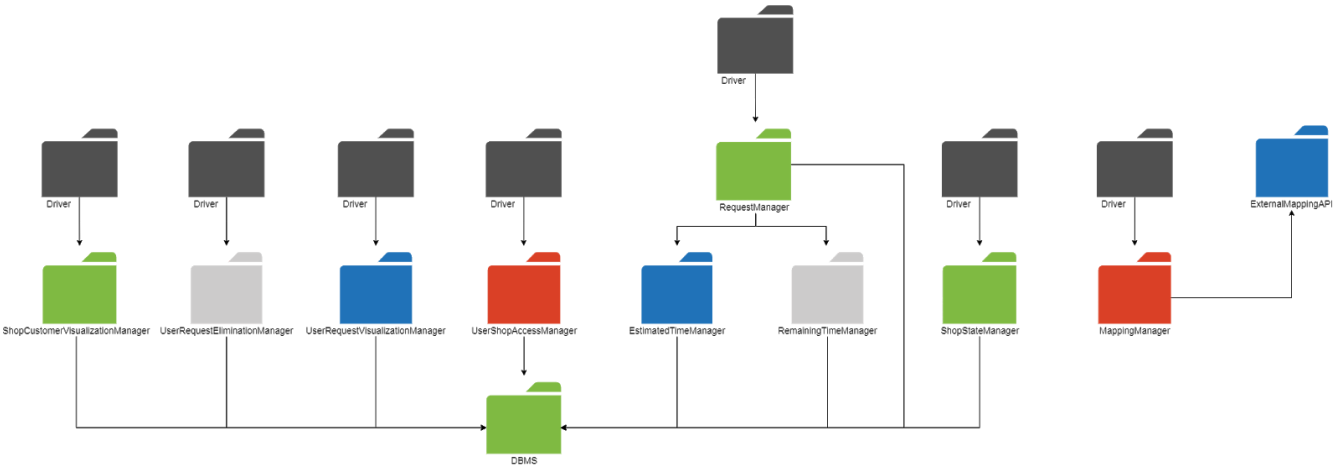
4. In the next step the fundamental RequestManager is implemented, unit-tested and integrated. A driver is needed to test his functionality in the sub-system.



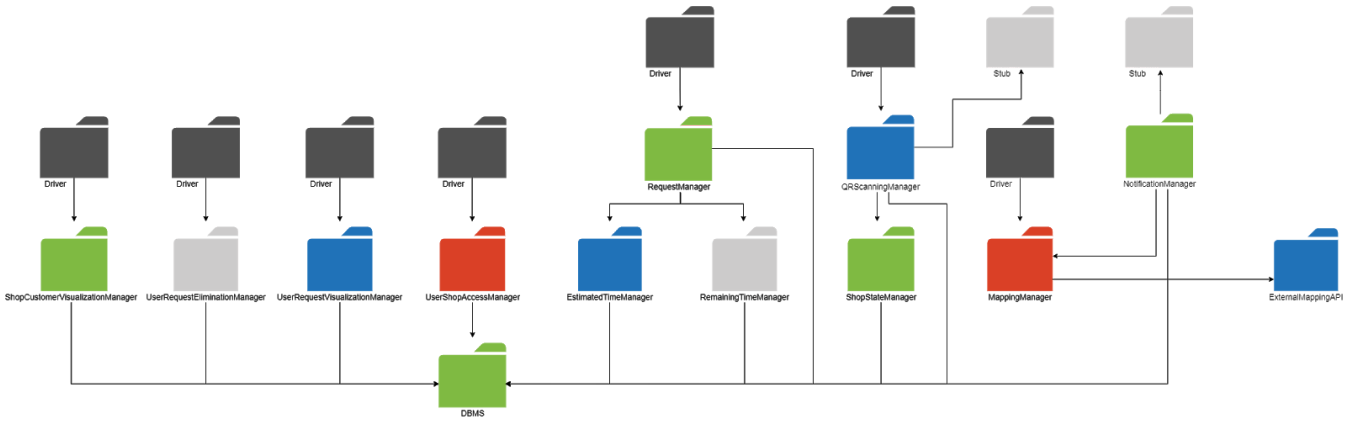
5. In the fifth step, are added to the system the UserRequestVisualizationManager, the UserRequestEliminationManager and the ShopCustomerVisualizationManager. Each of them is related to a specific driver that allows the related testing.



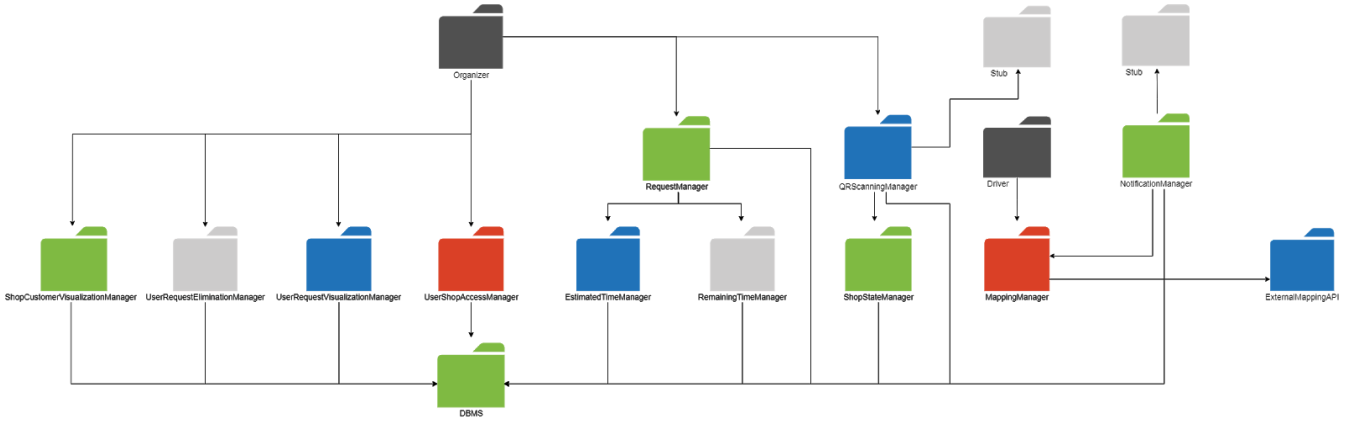
6. Then, it is possible to implement, integrate and test the ShopStateManager and the MappingManager. The latter will depend on the ExternalMappingAPI component that will be consequently integrated but neither implemented nor tested. It is assumed that it is offered by a reliable company.



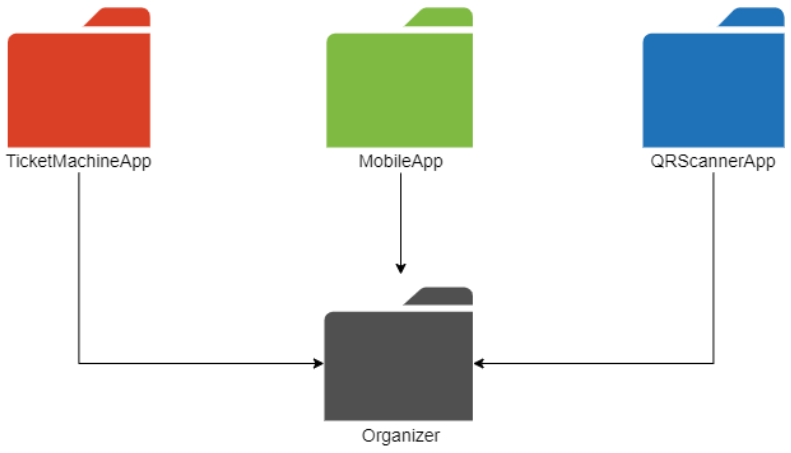
7. In the seventh step, the implementation and testing of the QRScanningManager and NotificationManager takes place. However, besides the normal driver, two stubs are needed to cope with the lack of communication from the NotificationManager and QRScanningManager to the MobileApp that has to be still integrated. The introduction of stubs is not typical of the Bottom-Up approach but is needed to perform testings. The MappingManager still needs a driver to test the interaction between it and the not yet implemented MobileApp.

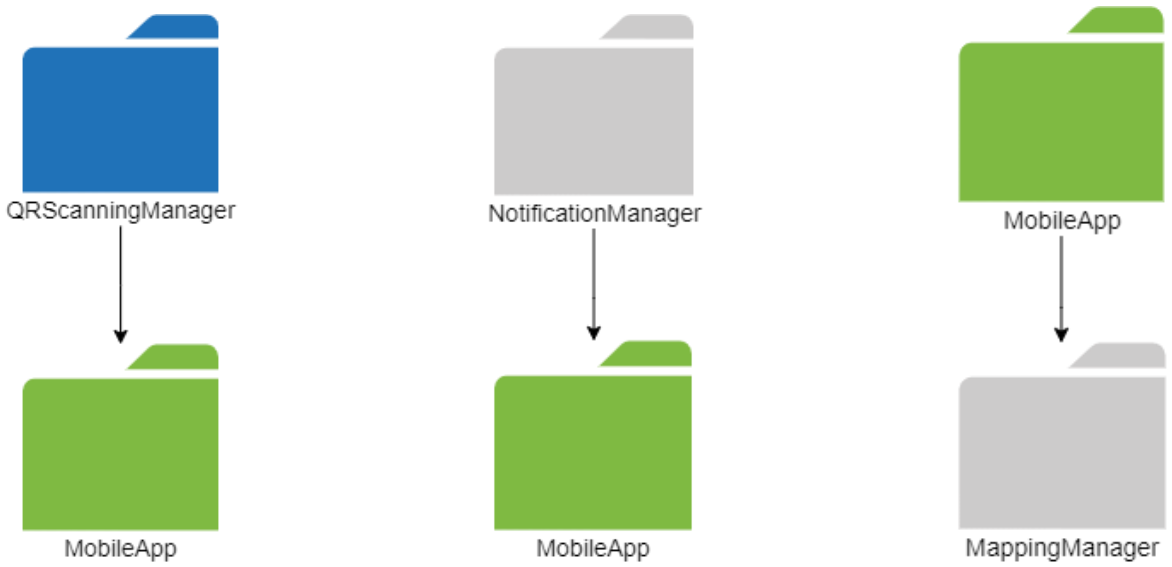


8. In the final step, the Organizer is implemented, integrated and tested and it substitutes the previous defined driver. It is not associated with the NotificationManager that communicates only with the MobileApp and MappingManager.



Finally, it is possible to implement, integrate and test the client side component. As well as relying on the Organizer, the MobileApp simultaneously provides its interface to the QRScanningManager and NotificationManager and exploits the MappingManager interface to insert the position using the GPS. All these interactions are integrated and tested.





Once every component has been tested and integrated, system testing can be performed.

5.3 System testing

The system testing is conducted on a completed integrated system to verify that the function and non - functional requirements are satisfied.

There are many types of system testing can be performed:

- Performance Testing: Its aim is to identify bottlenecks that can affect response time and establish a performance baseline in order to compare it with different versions of the same product or a different competitive product. As prerequisites we need an expected workload and an acceptable performance. As it is already specified in the RASD document, the system should be able to receive, check and save the sending request by the user in less than 10 second. Being a service that guarantees the safety of people, CLup has to be reactive and reliable. For these reasons it must be able to handle and serve a large number of users and shops simultaneously. In order to make the system efficient, it is necessary to distinguish the inefficient algorithms, possible query optimization and hardware|network issues.
- Load Testing: the purpose of this type of testing is to expose bugs, memory leaks and buffer overflow and so understand the load upper limit of the components. In order to reach this objective, it is necessary to increase the loud until the threshold and let operate the system with this load for a long period.
- Stress Testing: the objective is to make sure that the system recovers gracefully after a failure. To reach this critical condition, the system resources should be overwhelmed during the test. One idea could be doubling the baseline number for concurrent users/HTTPS connections.

Chapter 6

Effort spent

Member 1: Andrea Razza (968483):

Topic	Hours
Chapter 1 - Introduction	1h
2.2, 2.3 - Overview: High Level Component, Component View	20h
2.3, 2.4 - Deployment Diagram, RunTimeView	30h
2.5, 2.6, 2.7 - Component Interfaces, Selected architectural styles, Other design decision	5h
Chapter 3 - User interface design	10h
Chapter 4 - Requirements traceability	6h
5.1 - Implementation plan	1.5h
5.2, 5.3 - Integration strategy, System testing	6h

Member 2: Loris Panza (967915):

Topic	Hours
Chapter 1 - Introduction	1h
2.2, 2.3 - Overview: High Level Component, Component View	20h
2.3, 2.4 - Deployment Diagram, RunTimeView	30h
2.5, 2.6, 2.7 - Component Interfaces, Selected architectural styles, Other design decision	5h
Chapter 3 - User interface design	10h
Chapter 4 - Requirements traceability	6h
5.1 - Implementation plan	1.5h
5.2, 5.3 - Integration strategy, System testing	6h

Chapter 7

References

- All the diagrams have been made using “Visual Paradigm 16.2”.
- The Mock Ups have been developed using “PhotoShop”.