POLITECNICO DI MILANO

AA 2020-2021

Computer Science and Engineering

SOFTWARE ENGINEERING2

DD

Design Document

Team Members

ID	Surname	Name
10609805	De Berardinis	Alessia
10611044	Dragoni	Arianna

Prof.ssa Elisabetta Di Nitto

1. Introduction

1.1 Purpose

This document aims to provide a detailed representation of the software described in the RASD document. It goes deeper into details and shows an overview of the architecture of the CLup application.
It aids in the critical analysis of the problem and the proposed solution by creating an overall guidance for the programmers who will develop the application.
For this scope it is possible to find all the components of the system with the interactions among them, their interfaces, the runtime behavior and the deployment.
The design characteristics are provided together with the design patterns adopted and some algorithms which highlight the more critical aspects.
This document also gives a description of how the implementation, integration and testing plan will be like.

1.2 Scope

As previously shown in the RASD Document, the scope of CLup Application is preventing crowds and situations of possible danger during the period of Coronavirus emergency.
Store managers can regulate and monitor the affluence of people in the shop by adding their stores with all the related information to CLup. In this way the customers can do the shopping in a safer way with the possibility to:

- obtain a position in the queue of a shop by *taking the ticket* online for the current day. With this basic service the customer can check in any moment the status of the queue by receiving the remaining time he/she has to wait and the number of people above him/her. Moreover, periodical notifications are sent to him/her until his/her turn arrives.
- receive a time slot by *booking a visit* for the following days. In this case CLup Application gives the possibility to provide the categories of items the customer has the intention to buy and an estimation of the duration of the visit.

The system ideated prevents the customers from queuing in front of the buildings with other people and allows them to respect the distance rules imposed by the Government.
To make the lining up mechanism effective, when a customer makes a reservation (ticket or visit) the system generates a QR code which must be scanned at the entrances and exits of the shop. This allows to track the number of accesses in the store every day, information needed to build statistics and to better organize the store.
CLup application also involves people who are not able to use the necessary technology or do not own a device: they can directly go to the store and exploit the fallback option of handing out tickets on the spot. This is made possible through the presence of one or more totems around the building.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Customer**: who signs in the application with the purpose of doing shopping.
- **Store Manager**: who adds a store in the application and has the purpose of organizing it according to the new rules introduced to contrast the coronavirus pandemic.
- **User:** who signs in the application and uses the available services for him/her purposes. The user can be a customer or a store manager.
- **Demographic**: particular sector of population (children, seniors, adults, …).
- **Ticket:** the number received which corresponds to the position in the queue.
- **Totem**: multimedia structure which allows people who cannot use the application to take a ticket.
- **QR Scanner**: digital structure used by the customers to scan their tickets when they enter/exit the stores in order to make the shop managers monitor the affluence of people in their shop.
- **QR Code**: bidimensional matrix composed of black modules put in a square schema used to memorize information about a ticket.
- **Reservation**: is a general term used to indicate either a taken ticket or a booked visit.
- **Book a visit:** refers to the option for the user to book a visit in a selected shop for the following days.
- **Take a ticket:** refers to the option for the customer to have a position in the queue of a shop in the current day.
- **System**: it is the software that is the objective of the document.

1.3.2 Acronyms

- **RASD:** Requirement Analysis and Specification Document
- **DD:** Design Document
- **GPS:** Global Positioning System
- **UI:** User Interface
- **MVC:** Model View Controller
- **API:** Application Programming Interface
- **HTTP:** Hypertext Transfer Protocol
- **JSON:** JavaScript Object Notation
- **REST:** Representational State Transfer
- **RDB**: Relational Database
- **DBMS:** Database Management System
- **RDBMS:** Relational DBMS

1.3.3 Abbreviations

- **Gn:** n-th goal
- **Rn:** n-th requirement

1.4 Reference Documents

- available slides on Beep
- R&DD Assignment A. Y. 2020-2021

1.5 Document Structure

Chapter 1: the first chapter - which is an introduction of the design document - provides the purpose and scope of the document. It also includes the definitions, abbreviations and acronyms useful to help the reader to better understand what is written. It is also here that it is possible to check the structure of the document.

Chapter 2: it includes all the architectural choices taken for the development of the application by highlighting the components and interfaces of the system, presented both from a high level and detailed point of view (component view, deployment view, runtime view). All the design decisions are provided in this chapter, together with the patterns adopted.

Chapter 3: this chapter shows how the system will look like by providing the user interfaces of the main functionalities. Some of the mockups – which were presented in the RASD Document – are shown also in the Design Document.

Chapter 4: it maps the requirements and goals defined in the RASD document with the design components provided in the Design Document.

Chapter 5: this chapter provides the strategy used to decide the order of implementation, integration and testing of the subcomponents of the system. It also provides the type of system testing.

Chapter 6: it shows the effort (number of hours) spent by each component of the group for the draft of the document.

2. Architectural Design

2.1 Overview

The architecture of the application is a distributed one organized according to the three-tier architecture, that divides it into three logical and physical computing tiers:

Presentation tier: the presentation tier is the frontend layer and hosts the user interface. It is the communication tier of the application and consequently it manages the interactions of the end user with the system.
Its main purpose is to display information and highlight in a clear way the functions to the users.

Application tier
The application tier is also known as the business logic tier or middle tier and represents the core of the application.
Here information collected in the presentation tier is processed by handling business logic, that is a specific set of business rules.
It also moves and processes data between the two surrounding tiers: the presentation tier and the data tier cannot communicate directly with one another.

Data tier
The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored, managed and retrieved from a database or file system. It hosts the data (which are persistently stored in a DBMS) and it is invoked to provide one or more services.

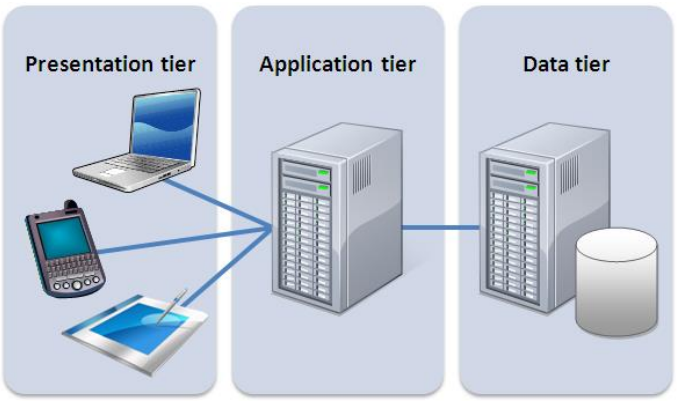


Figure 1: Three-tier Architecture

The hardware tiers described above are the machines on which the three logic layers that structure the application are subdivided: each logic layer (Presentation layer, Business Logic Layer, Data Access Layer) has its own dedicated hardware.

Three tier application was the prevailing one for client-server application because of its benefits. The main one is the fact that each tier runs on its own infrastructure, can be developed simultaneously by a separate development team, and can be updated or scaled as needed without impacting the other tiers.
Moreover, this kind of architecture allows interaction with the backends of many different applications despite having only one browser.

2.1.1 High-level components

The following diagram provides a high-level representation of the components of the system with their interactions also by highlighting the correspondence with the three Presentation, Application and Data levels.

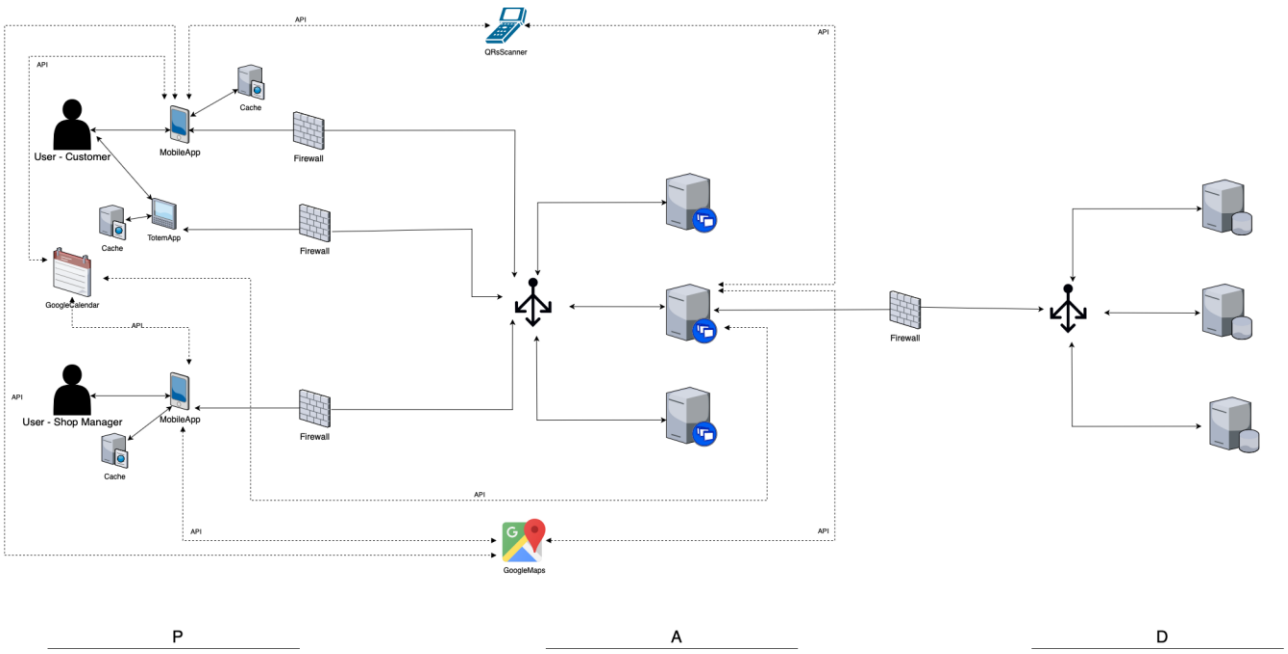


Figure 2: System Architecture

The presentation level is represented by the Mobile App and the Totem App that the users interact with in order to communicate with the Application level.
Specifically, the user customer can interact with both of them, while the user - shop manager only with the Mobile App.

The Application level is the main component, since it holds the business logic of the system: it interacts with the users and it has access to the database servers, where all the useful data is stored.

Moreover, in case of Mobile Apps, the Application Servers, which are the middle tiers, have access to the APIs of Maps (provided by Google Maps), Calendar (provided by Google Calendar) and QRsScanner.

To go into further details, the Presentation Layer exploits a Cache to make the requested data available when needed and to fasten the communication.

The communication between this layer and the Application one involves the use of a Firewall - which establishes a barrier between the internal network and incoming traffic from external sources - and of a Load Balancer. The latter is used to distribute the working load among the various Application Servers, that - due to security reasons and to improve reliability - are replicated. That is exactly why the Application Servers cannot exploit the use of caches.

The Business Logic layer communicates with the Data Access one both to retrieve information and to store data from/into the database. Also in this case and for the same reasons explained before, the Firewall and Load Balancer components are exploited. Also Database Servers are replicated to manage the risk of data loss and improve the safety of the system.

2.2 Component View

The Component Diagram below shows the main components of the system and the services provided by them. It captures the physical structure of the implementation and it is a static diagram since it describes several instances without explaining how they collaborate with each other but highlighting that at a certain point in time they will collaborate.

The focus is on the Application Layer - which contains the business logic - while the other two layers (Presentation and Data) are represented in a simplified way (as black boxes) only to show the interaction with the Application level.

Since the Totem is part of our system and we have the intention to implement it, it is represented as a Component in the diagram. Instead, the QR scanner is external and so we only include in our system the interface provided by it.

Regarding the Application level, the components proposed are the following ones:

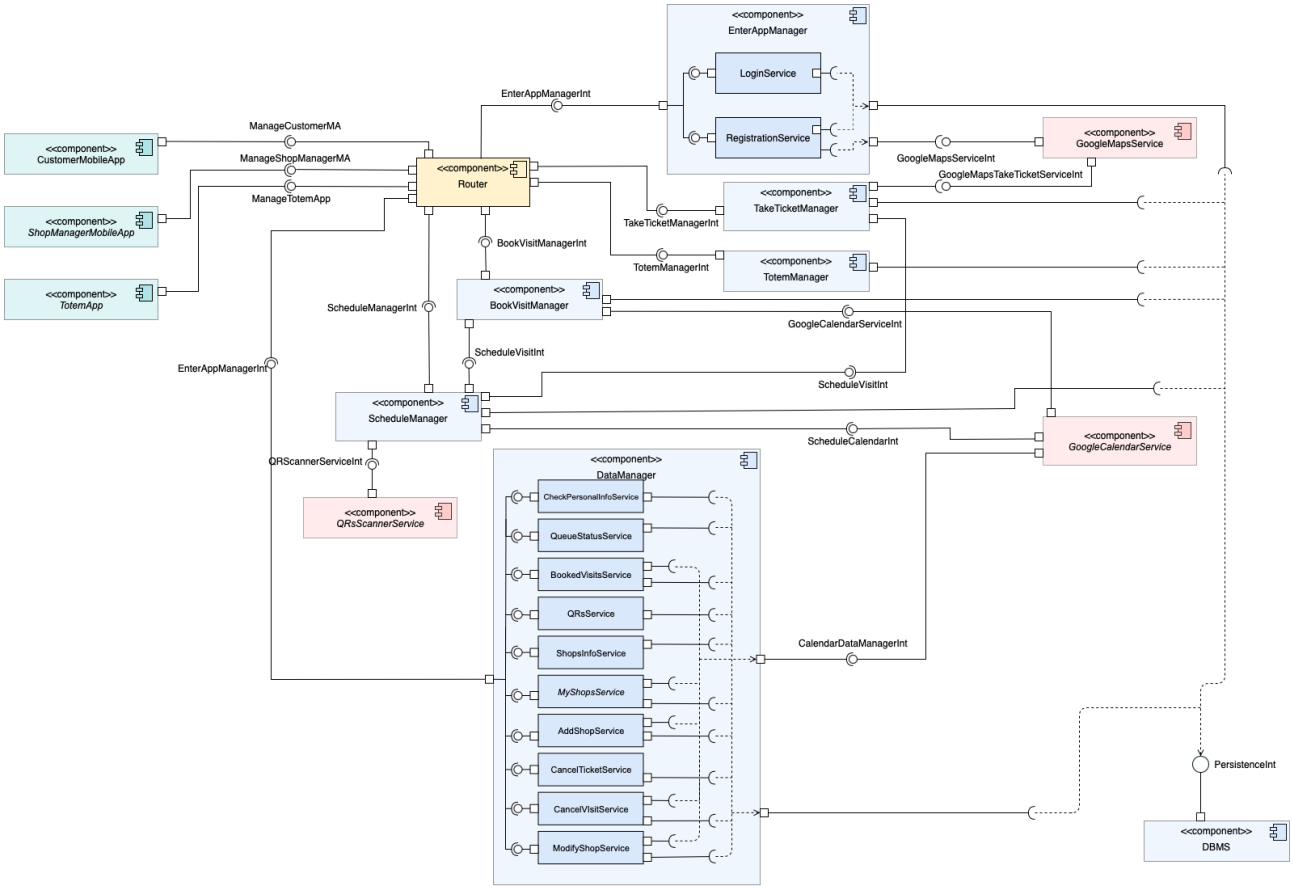


Figure 3: Component Diagram

- **Router:** this component takes the invocations from the clients and directs them to the right components of the Application Server. It is the dispatcher of the requests received and for this reason it must be always active.

- **EnterAppManager:** this component is used to allow the users to enter the application and exploit the services provided. It regards the Registration and the Login activities, and it interacts with the DBMS both in writing - to store data of the users in the database - and in reading - to check the correctness of the inserted values.

  The services provided are:
  - *LoginService*: it manages the authentication of a user to the application.
  - *RegistrationService*: it manages the registration of a user to the application

- **TakeTicketManager**: this component deals with the requests of the tickets done by the customer-users. It stores the data related to the ticket in the database and communicates with the ScheduleManager:
  - to get the queue corresponding to the selected shop
  - to check if the request of the ticket can be accepted or refused (in case the queue is full until the closing time, the user selects a shop when it is closed, a customer has already taken a ticket or booked a visit at the same time of the ticket).

- **TotemManager:** this component manages the requests of taking a ticket from the users who want to take it on the spot. It includes the printing of the QR codes.

- **BookVisitManager:** this component manages the requests of the reservations for a visit made by the customer-User. It saves the data related to the booked visit in the database and communicates with the ScheduleManager:
  - to identify the available time slots for the selected day for the visit
  - to decide whether to accept or refuse the request (rejected in case the user books two or more overlapping visits, takes a ticket which overlaps with the visit, selects a day that has not available time slots).

- **DataManager:** this component handles different requests made by the users concerning the visualization and/or modification of data. The related services are the following ones:

  - *CheckPersonalInfoService*: it allows the user to visualize the data related to his/her registration and to modify it.

  - *QueueStatusService:* it allows the user customer to check how many people are above him/her in the queue of the shop related to the taken ticket and the remaining waiting time

  - *BookedVisitsService:* it allows the user customer to check all the booked visits with the related QR codes.

  - *QRsService:* it allows the user customer to visualize all the QR codes related to the reservations done and to download them. It is also used during a “take a ticket” or a “book a visit” action if the customer decides to download the QR code of the reservation made.

  - *ShopsInfoService:* it allows the user customer to visualize the list of shops with the related information (how many people are already in the corresponding queue, which categories of items the store provides, the address and the opening and closing time). In case of GPS localization provided, it also sorts the list from the closest to the furthest shop.

  - *MyShopsService:* it allows the user shop manager to check and modify the information of the shops added by him/her. He/she also visualizes the daily reports.

- *AddShopService:* it allows the user shop manager to add a shop to CLup.

- *CancelTicketService:* it allows the user customer to cancel the taken ticket.

- *CancelVisitService:* it allows the user customer to cancel the booked visit.

- *ModifyShopService:* it allows the user shop manager to modify the information of one of his/her shop. It also includes the deletion of a store.

- **ScheduleManager:** this component has the purpose of managing the schedule of each inserted shop and, in case of Book a visit option, of computing the free slots of a selected day, by exploiting the Google Calendar API. It interacts with the DBMS in reading to get the data related to a reservation (also when needed by other components from the database) and to send periodical notification to the user customer about the reservations when it is almost his/her turn (in case of take a ticket) and two hours before (in case of Book a visit). For this scope the ScheduleManager interacts with the GoogleMapsService Component.

- **DBMS:** it allows all the other components to interact - to store, read, modify data - with the database. It must be always active to receive the requests of the components.

The external components of the system are the following ones:

- ***GoogleMapsService****:* it is the component that provides the interface through which the user can visualize the map of the city.
It is exploited by: EnterAppManager (to provide the localization during the registration phase) and TakeTicketManager (to compute the estimated time to reach the destination in case of localization provided).

- ***GoogleCalendarService****:* it is the component that provides the interface through which the user can visualize the days of the month.
It is exploited by: BookVisitManager (to allow the customer to select a day for the visit), DataManager (for BookedVisitsService, MyShopsService for the opening and closing time and the daily reports, AddShopService and ModifyShopService for the opening time slots, CancelVisitService) and ScheduleManager (for the daily entrances and exits).

- ***QRScannerService****:* it is the component that provides the interface required for managing the entrances and exits of customers from/to a shop.
For this purpose, it is exploited by the ScheduleManager

2.3 Deployment View

The deployment diagram above provides the structure of the run-time system and captures the hardware that will be used to implement it.

It shows the configuration of run time processing nodes and the components that live on them.

In this way it highlights a specific view - the deployment one - that shows the software in the context of the various hardware components.

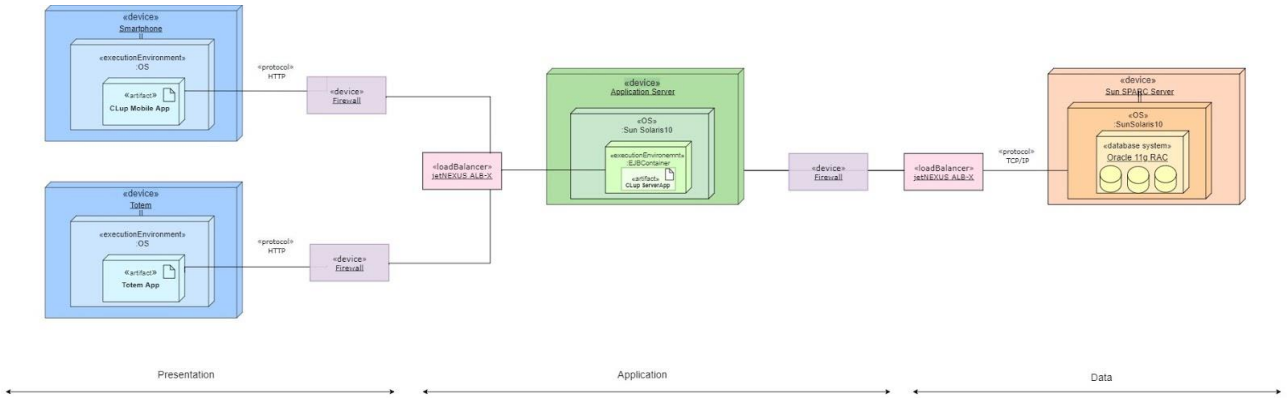


Figure 4: Deployment Diagram

*For simplicity we consider only the OS operating system in the diagram

CLup requires the deployment of software on the following nodes:

1. **Smartphone:** the users (both customer and store manager) must own a smartphone where to download the mobile application in order to exploit the services provided. The user customer needs it to use the basic functionality of taking a ticket and the advanced one of booking a visit. The user Shop Manager is required to have the device to add one or more shops to CLup.
   The mobile application must be available for both Android and iOS.

2. **Totem:** it is a multimedia structure which allows people who cannot use the application to take a ticket (and receive a printed copy of the QRcode) on the spot. It is used only by the User-customer.

3. **Firewall**: a firewall is a security device that can help to protect the network by filtering traffic, analyzing it and blocking outsiders from gaining unauthorized access to the private data.

4. **Load balancer:** is used to distribute network's traffic across multiple servers with the purpose of ensuring that a single server does not bear too much demand. It improves the application's responsiveness.

5. **Application Server:** it is a server designed to install, operate and host applications and associated services for end users. It handles the main business logic and processes the data sent from other servers. It is connected by involving the firewall to the MobileApp and to the TotemApp to answer the requests received by providing all the information needed. It interacts also with the Database server to get or store data from/in the database.
   Moreover, the application Server (only in case of Mobile App) has access to the APIs of the external components – Maps, Calendar and QR scanner - and is replicated to improve reliability and for this purpose the load balancer is used to divide the workload.

6. **Database Server:** it is a server aimed at providing services related to accessing and retrieving information from a database. It is used for storage and the data can be accessed through the execution of a query by using a query language specific to the database, like SQL.
   The Database Server is useful to deal with a lot of data, especially when there is the need to process it too frequently, as in the case of CLup application.
   The decision of using Oracle RAC comes from the several benefits it is able to provide. The main ones are the following:
   - *High availability.* It consists of ensuring that the data is not lost, and that the database remains in a normal state of operation by avoiding the loss in case of downtime.
   - *Easy scalability.* At the beginning the database environment is set according to the standard configuration to maintain a low cost during the initial stage and avoid unnecessary waste. When the system needs more processing power or has the necessity to increase the storage, it is possible to obtain a level of expansion without downtime.
   - *Reduced cost.* To improve the processing power of the system it is possible to add nodes with a cost which is lower than the cost of using a high-performance server.

2.4 Run Time View

The sequence diagrams provided above are useful to show the dynamic behavior of the system by highlighting its internal components (the external ones are not considered in this section). In addition to modeling the behaviors, the diagrams also model the flow of control and illustrate typical scenarios.

1. Login

This sequence diagram shows the Login made by a User (both customer and Shop Manager)

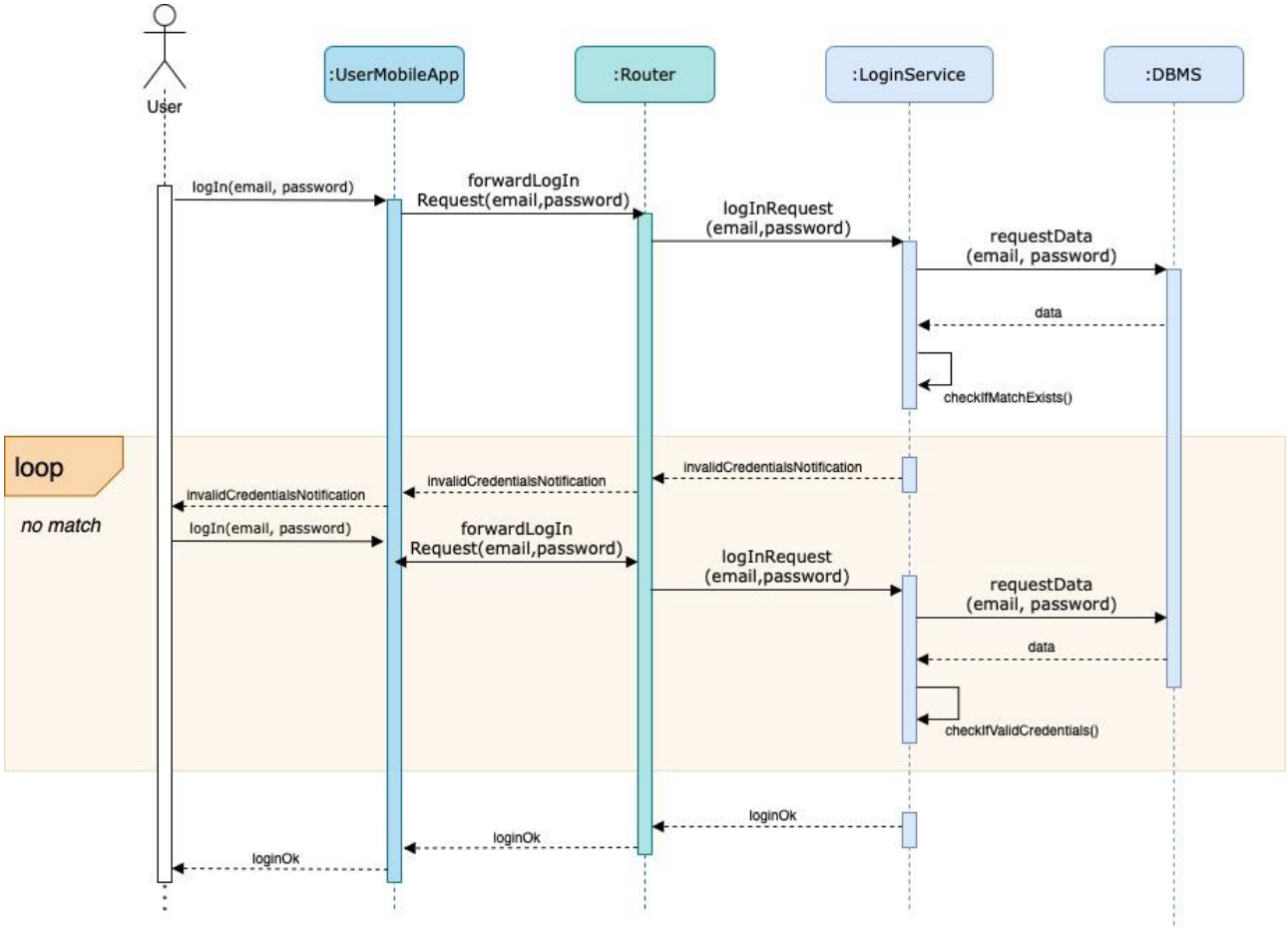


Figure 5: Login

The user opens the mobile application on his/her device and fills the mandatory fields with the email and password, already set during the registration.

After receiving the login request by the UserMobileApp, the Router sends it to the LoginService component, which manages the authentication of the user. To do this the component asks the data(email,password) of the user to the DBMS - who sends it back- and checks if there is a correspondence between the email and password inserted. If there is a match, the user can successfully login to CLup application, otherwise he/she has to try again.

2. Sign up User

The following two diagrams show the registration of the user.

The first one represents the Sign up of the User Customer.

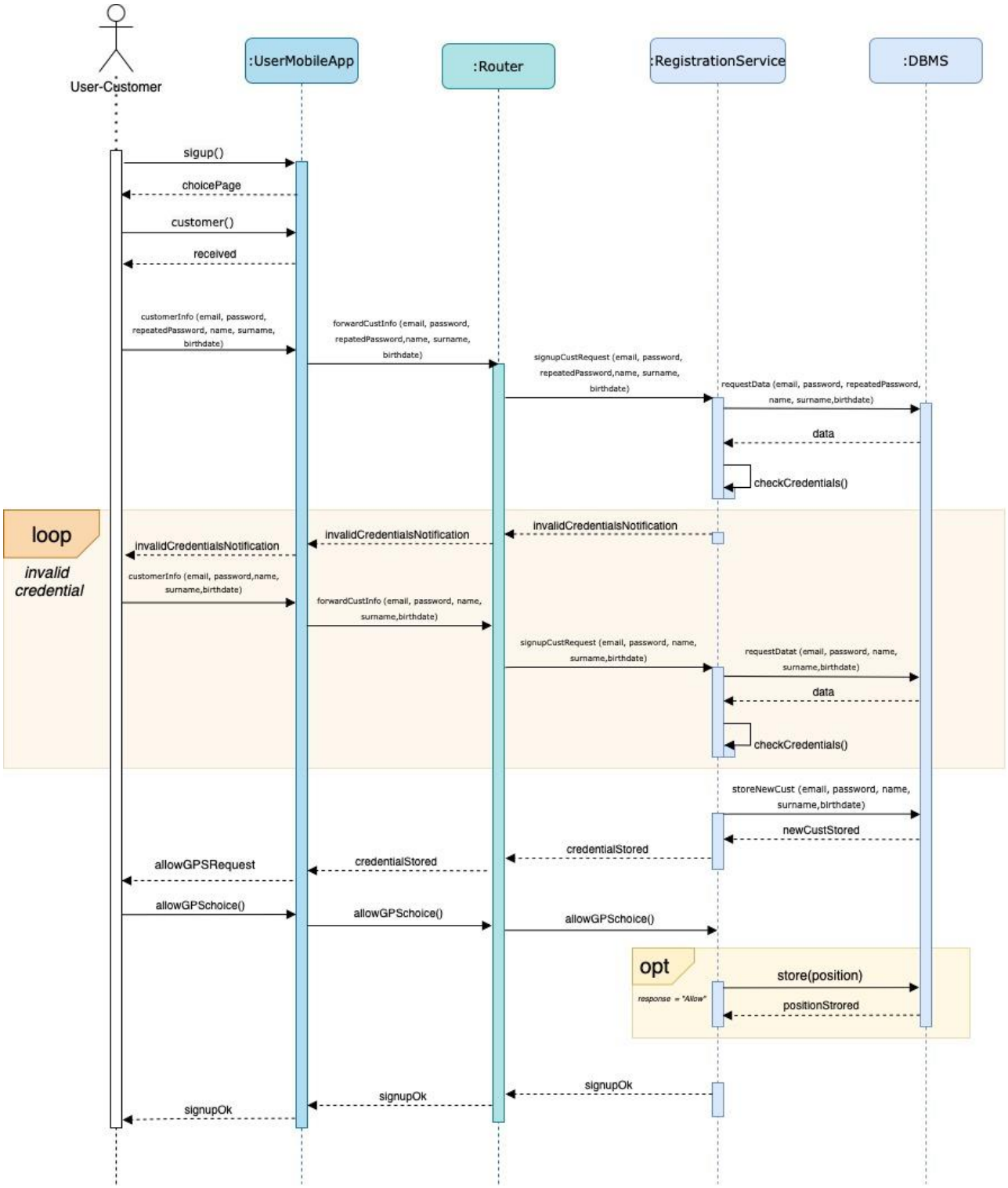


Figure 6: Sign up Customer

The user customer signs up through the mobile application. The MobileApp sends as response the page with the choice between “Customer” and “Shop Manager”. Once selected the “Customer” option, the request with the mandatory fields (email, password, Name, Surname) and the optional field (Birthdate) is forwarded by the MobileApp to the

RegistrationService (by passing through the router). To check if all the data inserted is valid, the RegistrationService component exploits the DBMS which sends it back the requested data.
-If the result is negative (invalid credentials) the user has to make the registration again.
-If the result is positive (*credentials ok)* the data is saved in the database and the RegistrationService component, after receiving the response from the DBMS, forwards it to the Router and User Mobile app, which sends the request of GPS localization to the customer who can decide whether to provide it or not (by clicking on “Allow” or “Do not allow” button). If the user gives access to it, the information is stored in the database.

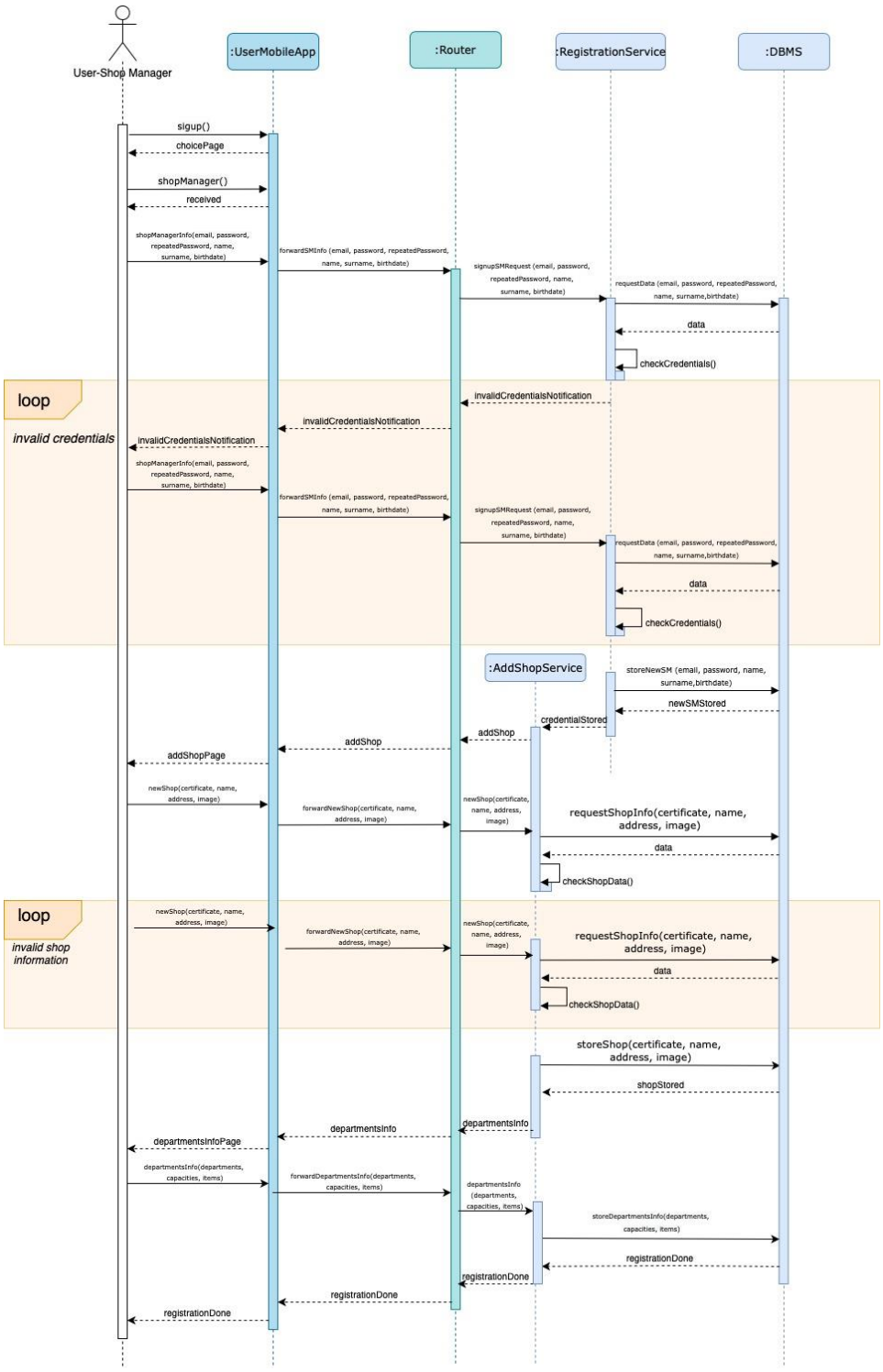


Figure 7: Sign up Shop Manager

This diagram in figure 7 shows instead the SignUp of the User-Shop Manager. As in the previous case the user, after selecting the “Shop Manager” option, fills the mandatory and optional fields about him/herself. After the credentials are checked through the DBMS, if they are correct, the RegistrationService component forwards the response received by the DBMS to the AddShopService component, which stores the information about the shop manager in the database and asks the user to add a Shop with the related information (certificate, name, address, picture) about it. This data is sent back to the addShopService (passing through the mobile app and router), which creates a shop and asks the Shop information to DBMS to check whether the shop already exists and if the certificate is valid.
If everything is correct, data is stored in the database and the addShopService component sends the request of inserting Departments’ information (capacities, items) to the user, which provides it. If the information of the shop is invalid, the user has to try again. The same happens if the credentials inserted by the user are not valid.

3. Take a ticket

The sequence diagram in figure 8 shows the functionality of taking a ticket. For simplicity we have considered the case in which the user has already provided the GPS localization during the registration phase.

The user opens the mobile application and selects a shop from the list of the available ones. The selection is forwarded to the UserMobileApp, Router, TaketTicketManager until it reaches the DBMS which sends the data related to the shop (people already in the queue, category of items, address, opening and closing time) back to the user taking it from the database.

At that point the user selects the “Take a ticket” option and this choice is forwarded to the TakeTicketManager which interacts with the ScheduleManager that invokes the DBMS to get information about the queue and the user’s reservations from the database. This information is forwarded back to the TakeTicketManager which decides whether to accept the request of the ticket or not:

- If yes, the TicketManager stores the information related to the ticket taken including the generated QRcode into the database and computes the time the customer needs to reach the destination and the people before him/her in the queue and sends them to the user. If he/she wants he can download the QR code generated: the user sends the request to the router which forwards it to the QRsService component which asks the DBMS to get the requested QR from the database.
- If not, the request is denied, and an error message is sent to the user.

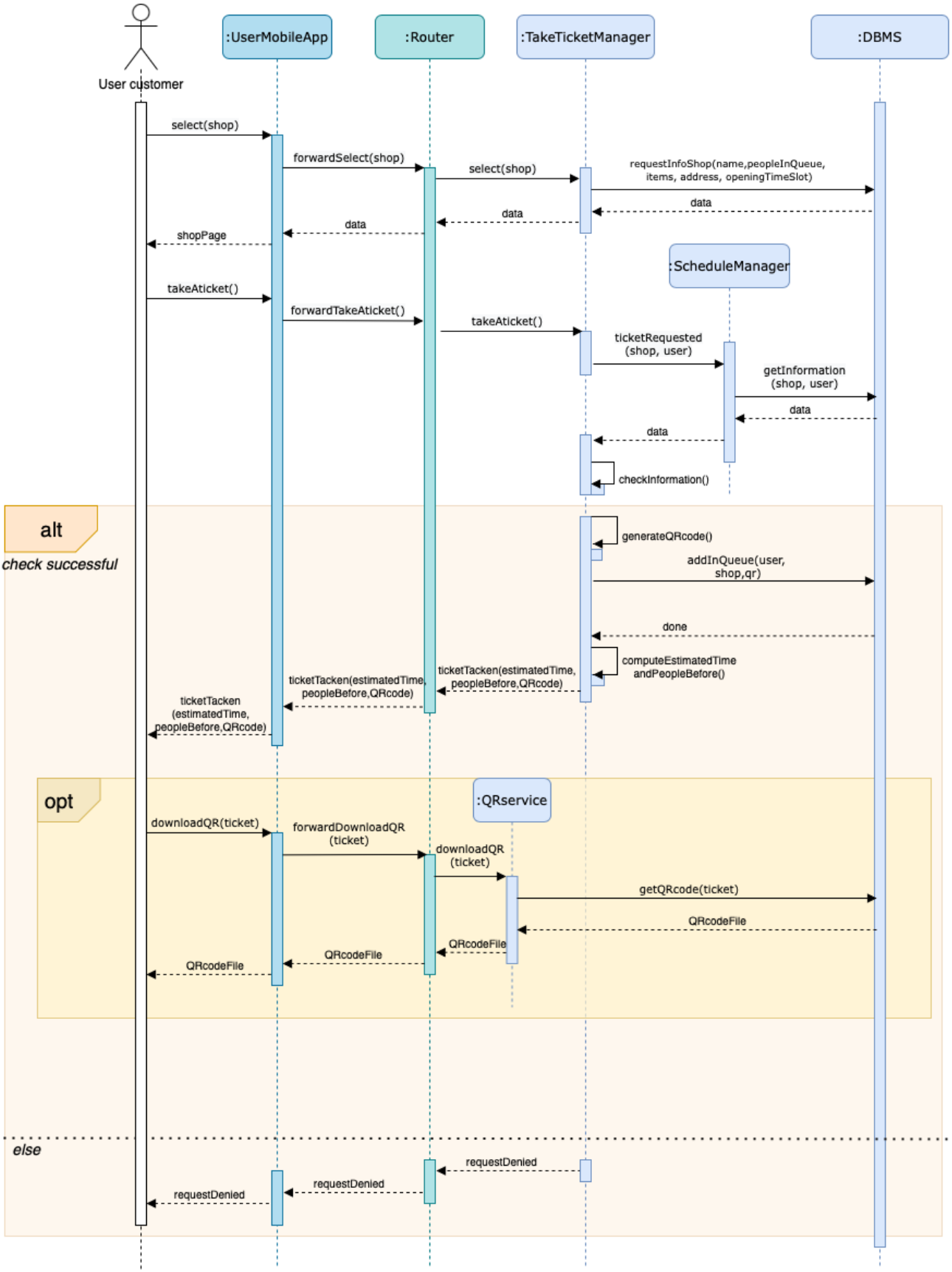


Figure 8: Take a Ticket

4. Book a Visit

The sequence diagram above shows the advanced functionality of booking a visit.

For ease we consider that: the day selected by the user has at least one available slot time (otherwise he would select another day).

As in the case of Take a Ticket option, the user selects a shop and receives the information related to it (name, people in the queue, items, address, opening and closing time) thanks to the BookVisitManager that gets the necessary information from the database through the DBMS.

Then the user selects the “Book a Visit” option with the day he/she prefers. This choice is forwarded to the ScheduleManager which receives from the database the schedule related to the selected day and computes the free slots.
The slot selected by the user is forwarded to the BookVisitManager which evaluates the request of reservation and decides whether to accept it or not:

- If yes, the visit is saved by the BookVisitManager in the database and the user is asked to fill the optional fields about the categories of items he/she wants to buy and the duration of the visit. The information is received by the BookVisitManager which generates the QR code which is stored in the database with the optional data thanks to the DBMS. As in the previous case the user can download the QR code. At this point the visit is successfully booked and the user receives the recap with the day, time, shop and QR code.

- If not, the request is denied, and an error message is sent to the user.

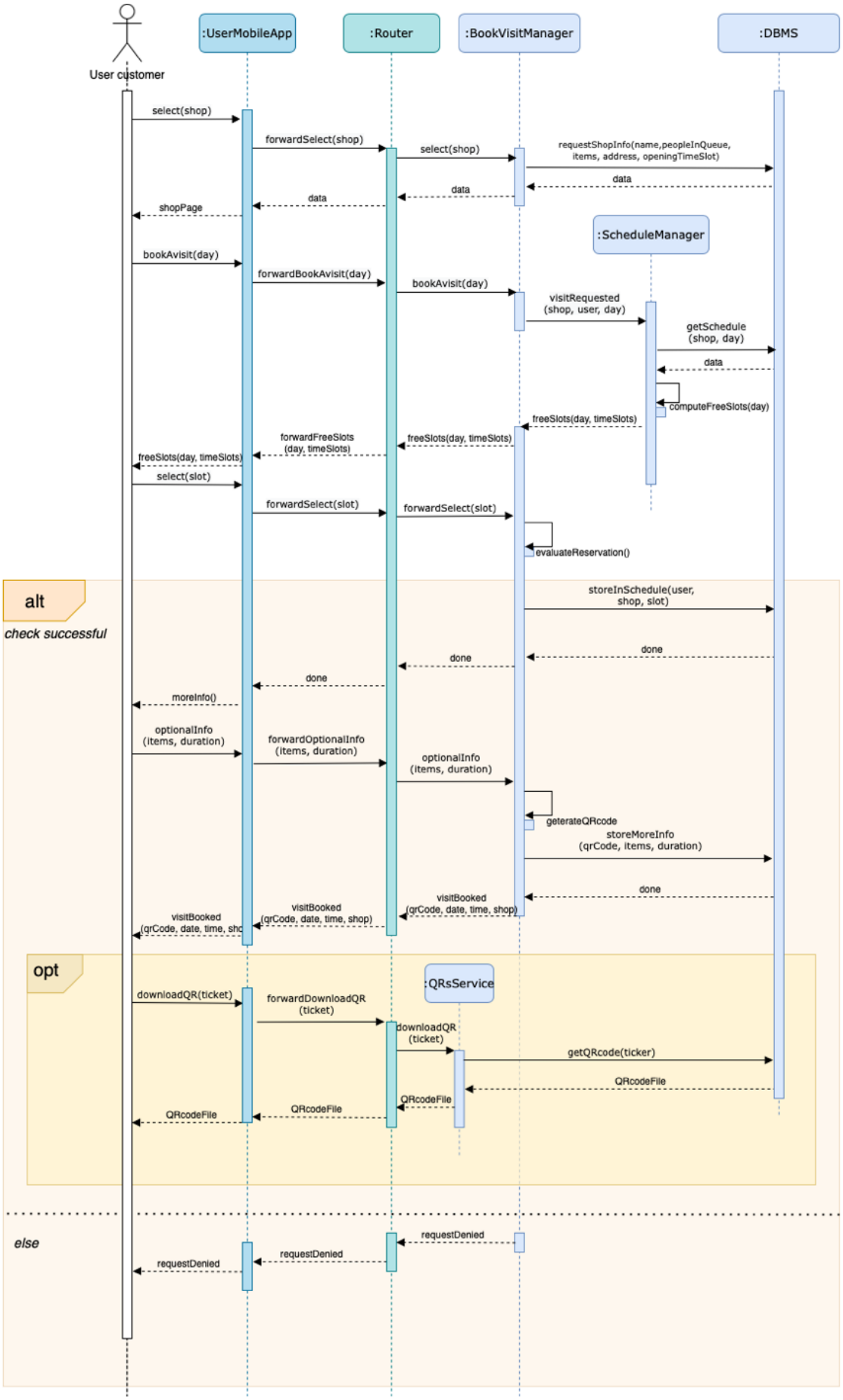


Figure 9: Book a Visit

5. Take a Ticket on Spot

This sequence diagram shows the functionality of taking a ticket on the spot, by exploiting the totem placed around a shop.

The user clicks on the “Take a ticket” option and the request is forwarded to the TotemManager, which - through the Schedule Manager - receives the necessary data from the database.
With all the information needed the TotemManager decides whether it can accept or not the ticket, by checking if the queue is full until the closing time of the shop. If the check is successful, the TotemManager, after generating the QR code, interacts with the DBMS to save the ticket in the database.
Then it computes and sends to the customer the time he/she has to wait and the number of people above him/her in the queue together with the generated QR code.
When the ticketTaken message is received, the totem prints it.
Instead, if the system refuses the ticket, the request is denied, and a message is shown to the customer.

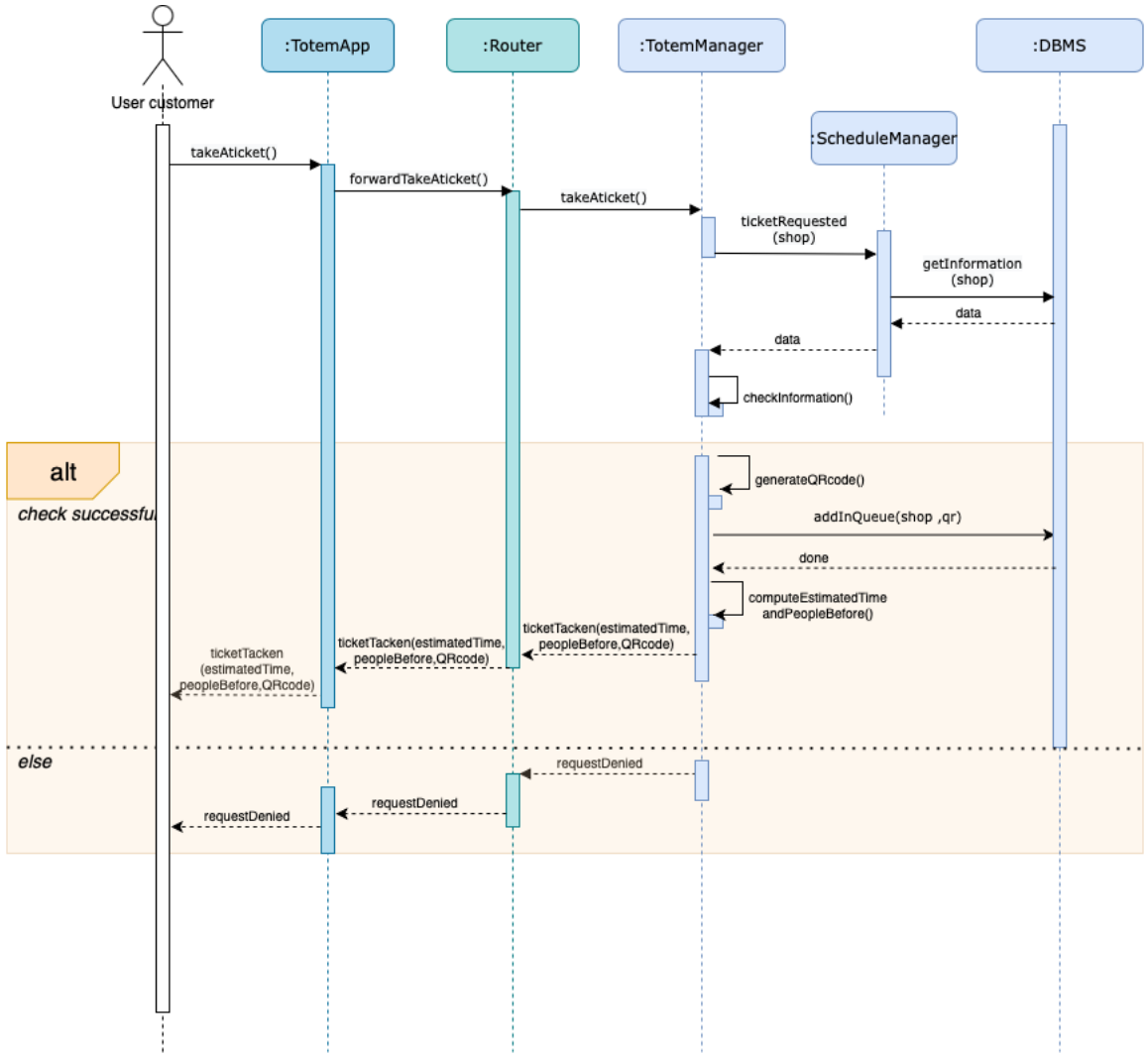


Figure 10: Take a Ticket on Spot

6. Check Information

The following sequence diagrams show the actions that a user can do through CLup application. We provide some possible combinations as examples.

6.1 The shop manager checks the list of his/her shops and modifies one of them

The user enters the application and checks the list of his/her shops, which is computed - for instance by showing the shops from the most attended one - and provided by the MyShopsService component. Together with the shop the user visualizes the number of people that have accessed it until that moment, information that MyShopsService component receives from the DBMS after requesting it.
The user's selection of the shop he/she wants to check is sent to the MobileApp and then forwarded to the Router and MyShopsService. The latter gets the details related to the shop from the database through the DBMS, computes the daily reports of the current and past days and sends the information to the shop manager.
Then the user can edit the information related to the shop: he/she can modify the name, the departments, categories of items and capacities but he cannot do the same for the address (in case of transfer the user must directly add a new shop). The request for editing is sent to the ModifyShopService, which is also responsible for storing the new modified information into the database.

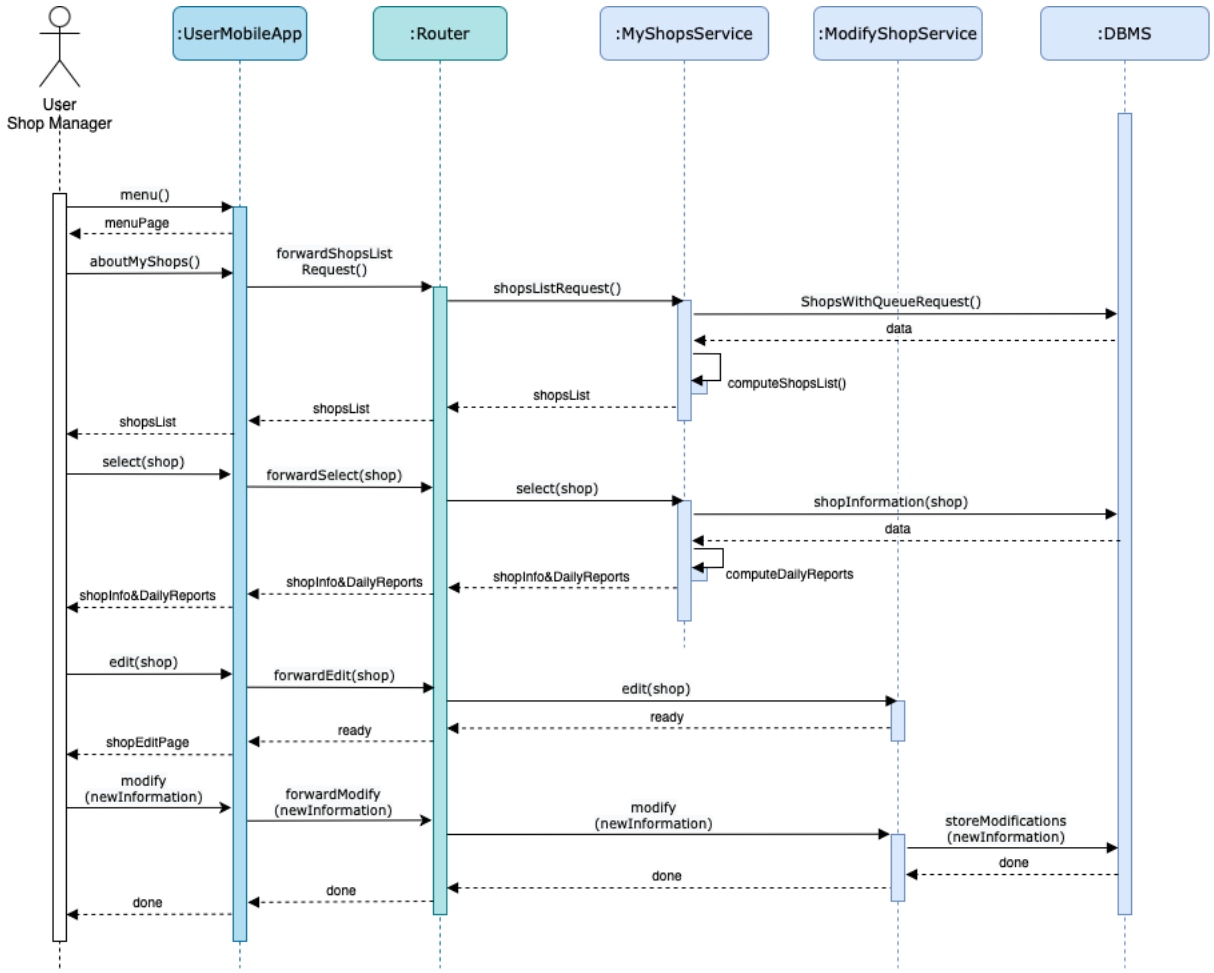


Figure 11: Check Information (1)

6.2 The customer checks the list of the shops with the related information, the visits he/she has already booked and decides to delete one of them

The customer opens the Application where he can *see the list of the shops*. The ShopsInfoService gets the list of shops with the related information from the DBMS and -in case the customer had already provided his/her GPS localization – sorts the list from the closest to the furthest one and sends it to the user. The information he/she can visualize are the number of people that are already in the corresponding queue, which categories of items the store provides, the address and the opening and closing time of the store.

Then the user decides to *check the visits he/she has already booked*. For this purpose, the request is sent to the BookedVisitsService which retrieves the list of visits from the DBMS and sends them to the user. Then he/she selects one of the visits to check the information (name of the Shop, the Date, slot time, the duration of the visit and the category of items he/she wants to buy (if provided)), that are got from the BookedVisitsService from the DBMS.

The system also shows to the customer the “CancelVisit” button. The request is forwarded to the CancelVisitManager that sends back to the user the Confirmation of the deletion action. Only in case the answer is yes, the CancelVisitService interacts with the DBMS to cancel the tuple related to the reservation from the database.

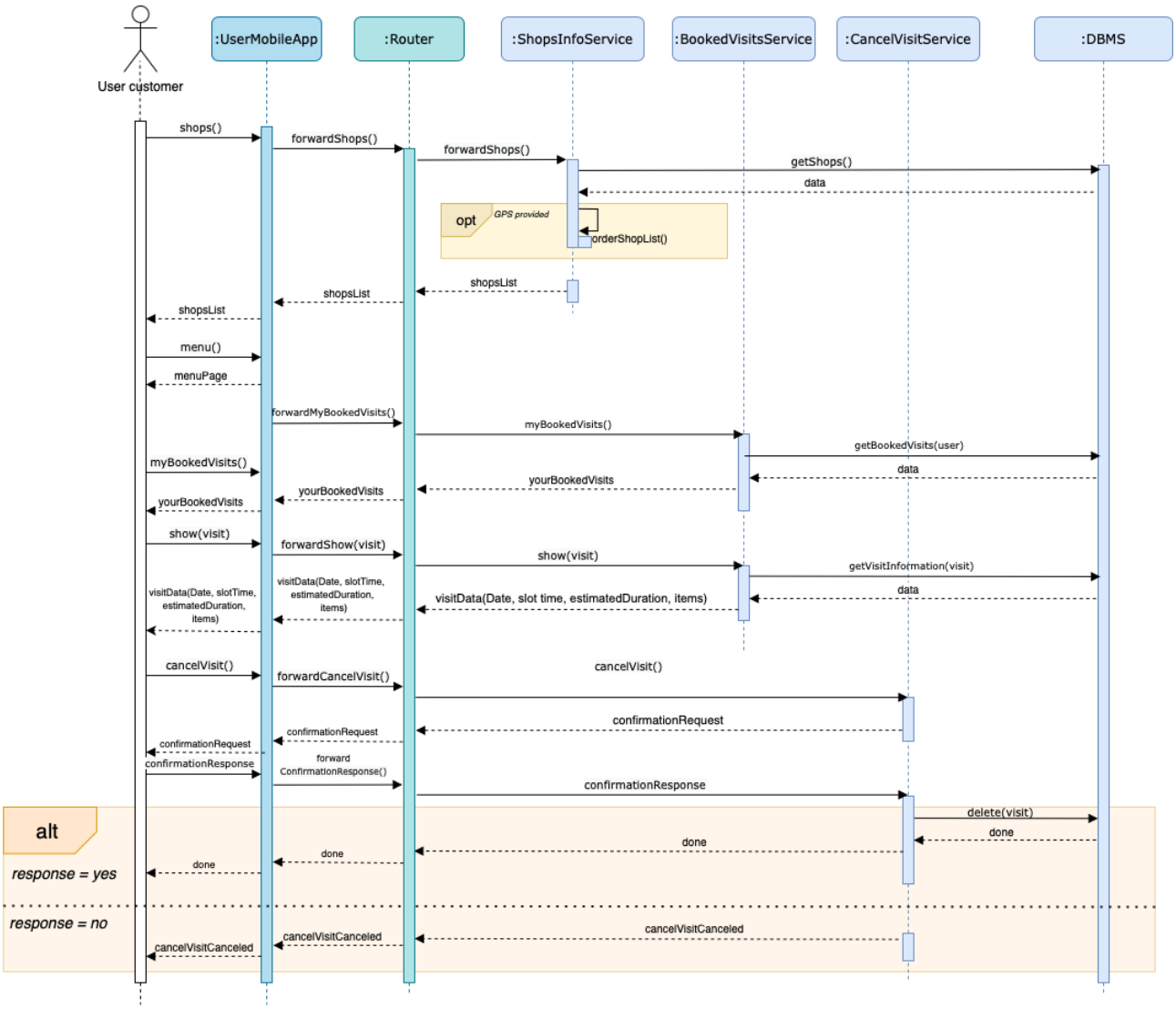


Figure 12: Check Information (2)

6.3 The customer sees his/her personal information, checks the status of the queue and deletes his/her reservation (ticket)

As first the customer exploits the MobileApp to see his/her *personal information*, inserted during the registration phase. This data is provided to the customer by the CheckPersonalInfoService which requests and gets it to/from the DBMS. Then the user exploits the functionality of checking the *status of the queue*. For this purpose, the QueueStatus Manager gets the necessary data from the DBMS and computes the number of people above the user in the queue and the remaining time he/she has to wait. This information is forwarded to the router, to the UserCustomerMobileApp and then to the customer. The latter is shown with this data also the CancelTicketButton. If he/she decides to *cancel his/her reservation*, the decision arrives to the CancelTicketManager that interacts with the DBMS to delete the ticket related to the user.

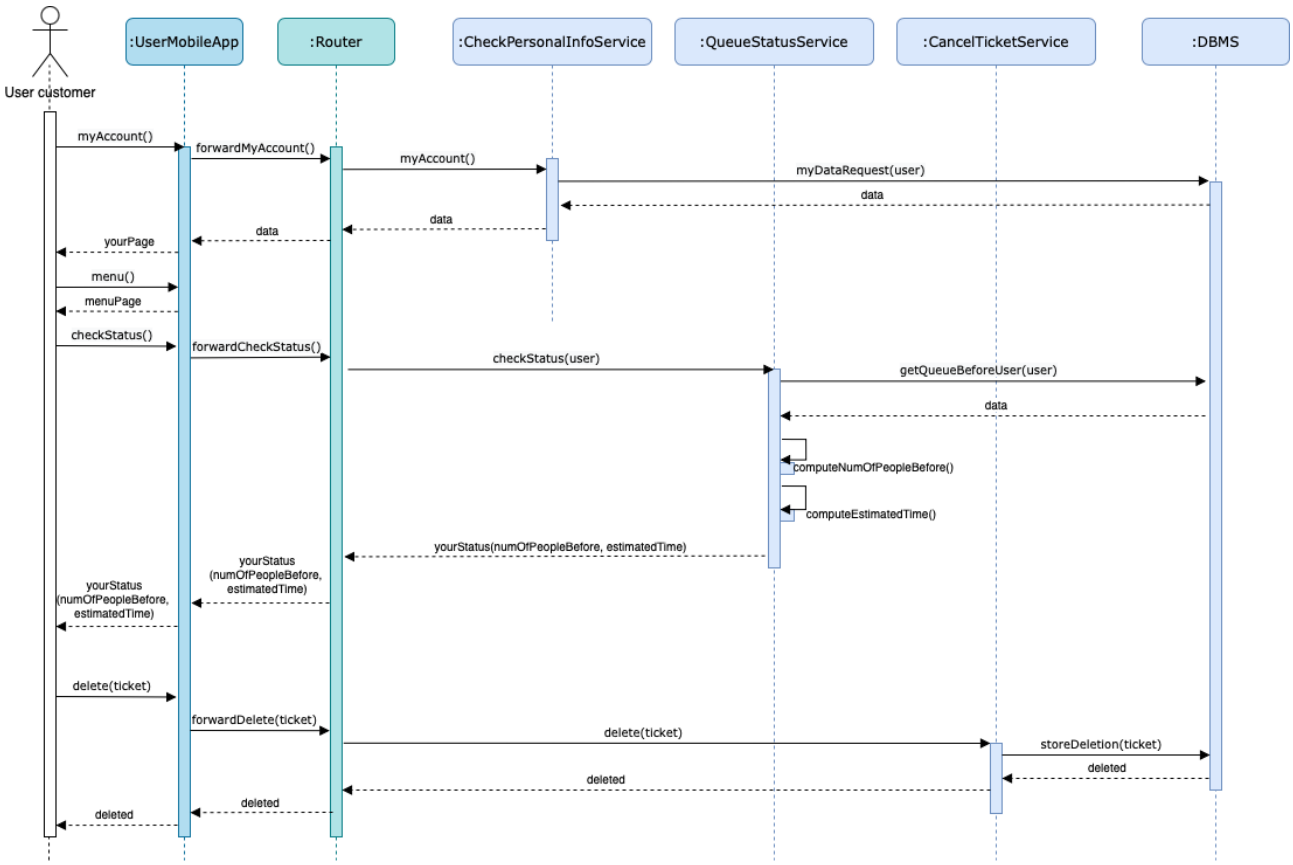
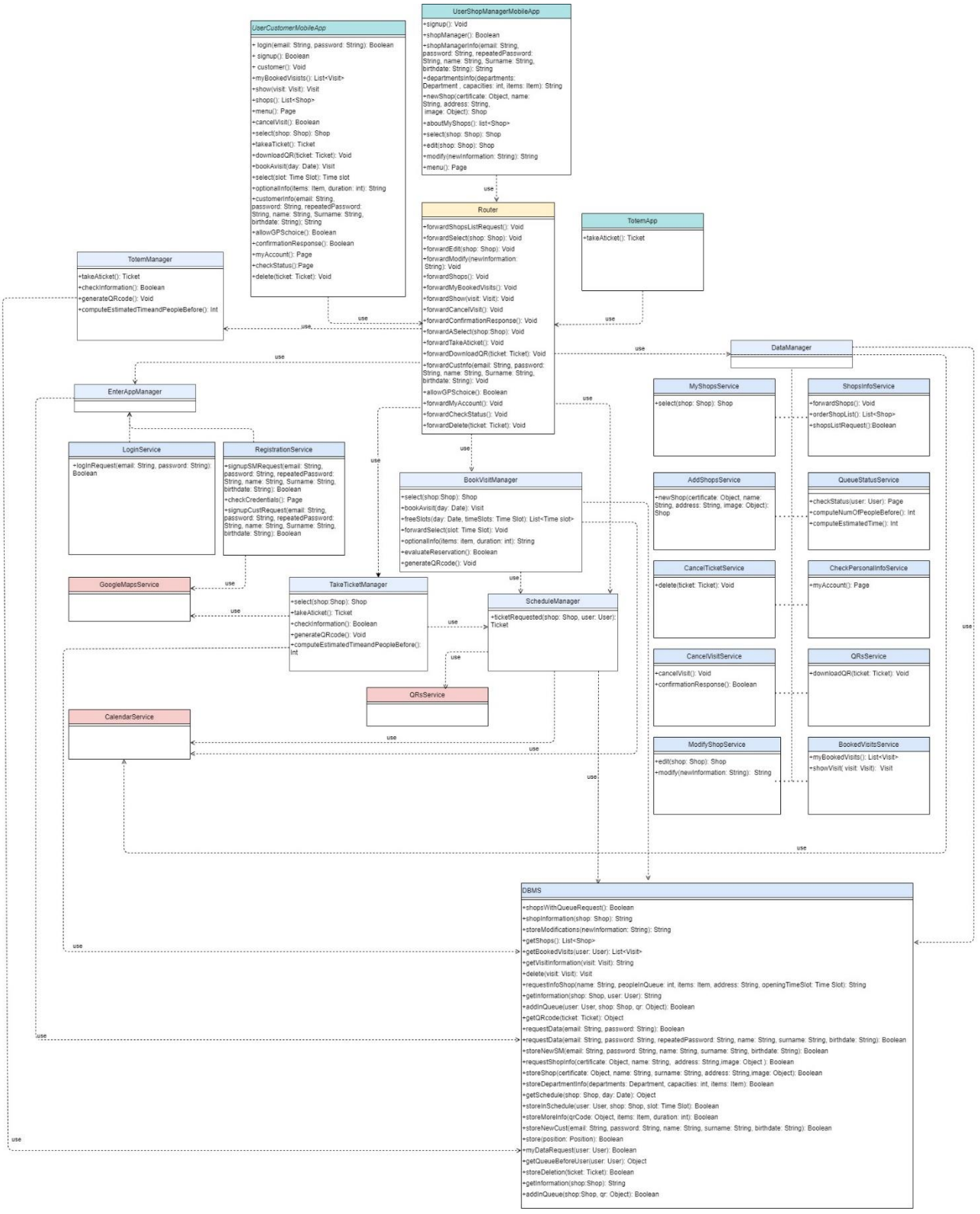


Figure 13: Check Information (3)

2.5 Component Interfaces

The next diagram provides the operations that a component can offer by highlighting the main methods that can be invoked. It also shows which are the dependencies between the various components. They are the same described in the section Component Diagram.



2.6 Selected architectural styles and patterns

2.6.1 Architectural styles

Three tier architecture
As previously mentioned, we have decided to use a three-tier architecture. It is a modular client-server architecture that consists of three tiers:

- the Presentation tier, which displays information of the services
- the Application tier, which handles logic,
- the Data tier, which stores information.

The decision comes from the several benefits that this type of architecture can give. In fact it is able to improve performance, reliability and availability.
The three-tier application is an optimal solution also because it would make things easy in case CLup evolved in time. For this purpose, it gives the ability to update the technology stack of one tier, without influencing other areas of the application and provides an ease of maintenance since it is possible to manage presentation code and business logic separately without impacting on each other.

JSON format
The format of the data that is exchanged between the components is JSON, a text-based data format which follows JavaScript object syntax. Since data are transmitted across the network, Json is useful because it can exist as a string.
It is also capable of providing to the developers a human-readable storage of data that can be easily accessed.

HTTP Protocol
In the system described, clients and servers communicate by exchanging messages and for this purpose the HTML protocol has been used: the messages sent by the client are called *requests* and the messages from the server are called *responses.*

HTTP is an application layer protocol which is sent over a TLS-encrypted TCP connection to provide secure data transfer. In fact, TLS is a data encryption technology that encrypts all data sent from one system to another with the capacity of preventing eavesdropping. By enabling the authentication of digital entities TLS helps to protect the application from cyberattacks.

APIs - REST
In the context of Google APIs - Google Maps and Google Calendar - the software architectural style used is REST, which with some HTTP verbs is able to retrieve and modify representations of data stored by Google. The fact that REST services use HTTPS means that they can be consumed by almost any ‘online’ device or application.
Moreover, REST is used in our application because it allows to achieve the architectural properties of performance, scalability, simplicity, modifiability and extensibility.

2.6.2 Pattern

MVC- Model View Controller

The pattern adopted to develop CLup Application is the MVC one. It is an architectural pattern that separates the data model of an application from the graphical representation (view) and from the control logic (controller), components that are built to handle some specific development aspects.

The *model* deals with data and the logic of the system by defining where the application's data objects are stored. It is connected to the database and is responsible for adding or retrieving data. Since the controller never talks with the database, it is up to the model to take the data from it and answer to the requests of the controller by sending data to it. The controller is the intermediary that enables the interconnection between the views and the model and after receiving data from the model, it processes it and sends it to the view: it controls the data flow into a model object and updates the view whenever data changes.
The latter only displays data: it creates an interface to show the actual output to the user by speaking with the controller to take the needed data (which are managed by the model component).

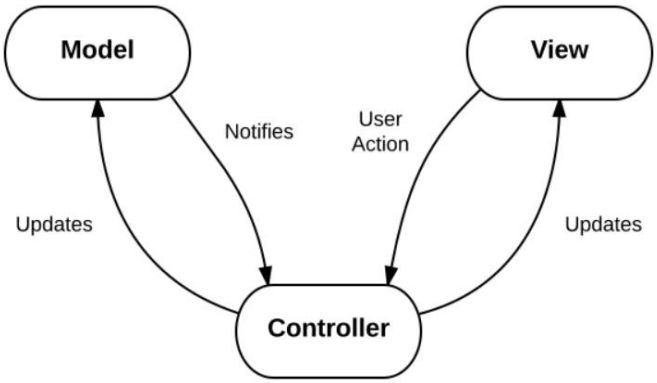


Figure 14: MVC

The decision of using the MVC pattern comes from the several benefits it can provide. The main one is that the division enables readability and modularity: it allows to design, implement, and test each portion separately by keeping the code organized. This helps to find what is needed quickly and to add functionalities with ease. So, thanks to this pattern it is possible to write efficient code and to re-use it.

Moreover, since CLup application can be affected by frequent changes, the adoption of this pattern suits it because it provides an architecture which is very change tolerable (thanks to the fact that it isolates the model part from the view one). This means that any modification in the model component does not have any significant impact on the whole architecture.

Observer Pattern

To implement the MVC pattern we have decided to use the Observer pattern, which has the aim to relay any state of change to an object in the quickest and simplest way.
The observer object has to register itself to the list of the observers of a subject, that informs the registered observers when a change occurs. The observable is the subject whose state may be of interest.
In our specific case:

- the controller is an observer of the model (and the model is the observable)
- the view is an observable of the controller (and the controller is the observer)

2.7 Other design decisions

Relational Database

The type of database we have decided to adopt is the relational database, which uses a structure that allows to identify and access data *in relation* to another piece of data in the database.
It maintains data in tables by providing an efficient and flexible way to store and access structured information.

This type of database has been chosen since it is possible to store, categorize and query data easily without reorganizing the tables of the database. RDB suits our application for the accuracy and consistency it is able to provide thanks to the strong validity checks performed.

It also improves the security characteristic of the system since data in tables within a relational DBMS can limit the access to specific users which is preferable that they do not access the data.

Moreover, since we predict frequent updates and additions of data, the relational database is an optimal solution because it is able to improve the scalability of the system.

Thin client

As mentioned in the precedent sections, most of the business logic is implemented on the server and not on the client side. For this reason, our decision is to use a thin client approach, which relies on the server to perform the processing of data.

The thin client fits our application since it improves security: in fact, it cannot run unauthorized software and data can't be copied or saved anywhere except for the server.

Other positive aspects that have led us to choose this approach are the capability of reducing costs and the ability of improving scalability. Also a more efficient manageability is provided.

Synchronous communication

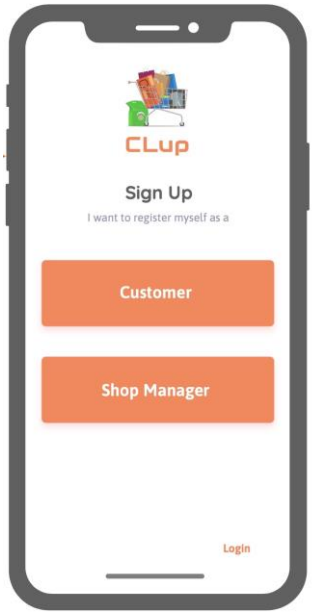
The type of communication adopted is the synchronous one that offers a real time communication between the components.

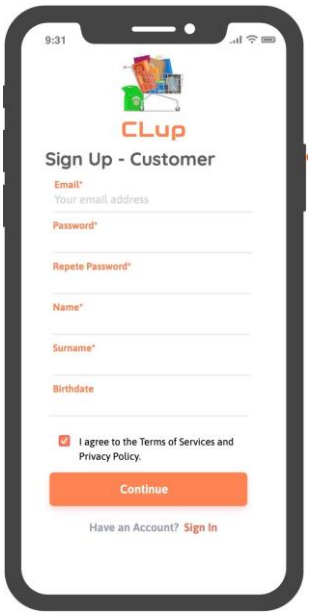
Each message which is sent requires a reply and is instantaneous: it is faster and more dynamic. We have decided to use this type of communication to make the components interact since it gives the opportunity for more in-depth interactions.
Since in our system it is necessary to transfer a large amount of data, synchronous communication suits this purpose with an effective and reliable transmission.

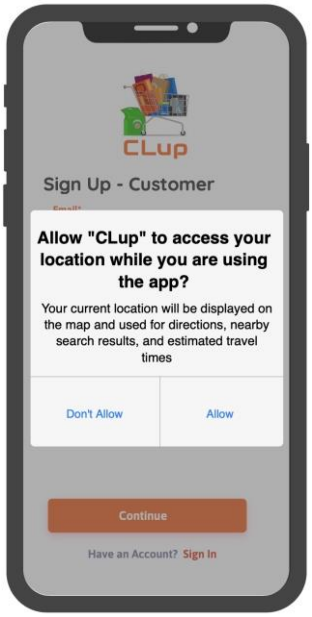
3. User Interface Design

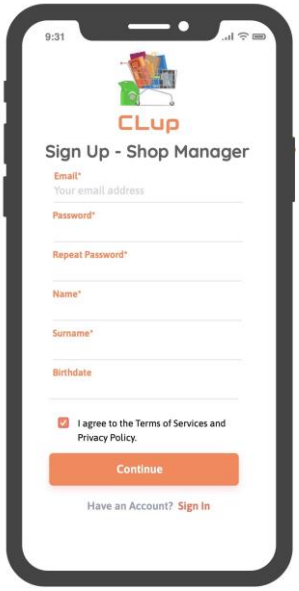
This section contains the main mockups which show how the application will look like. Some of them can also be found in the RASD document.

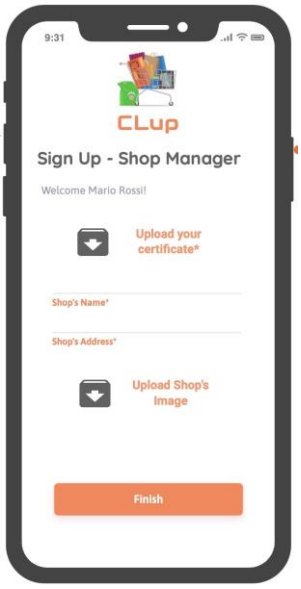
Signup Customer and Shop Manager

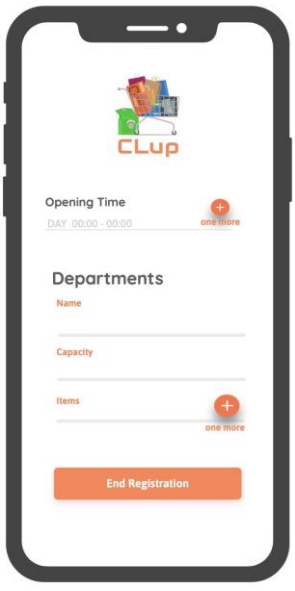




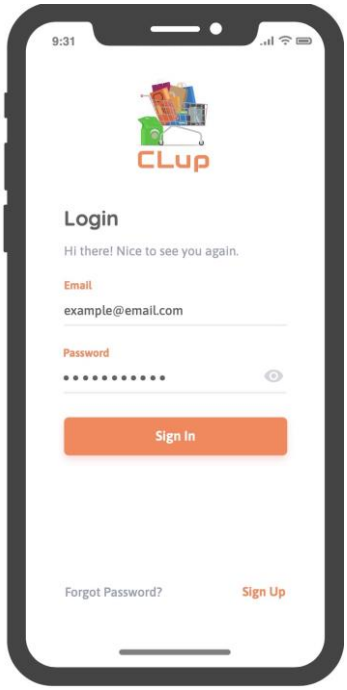




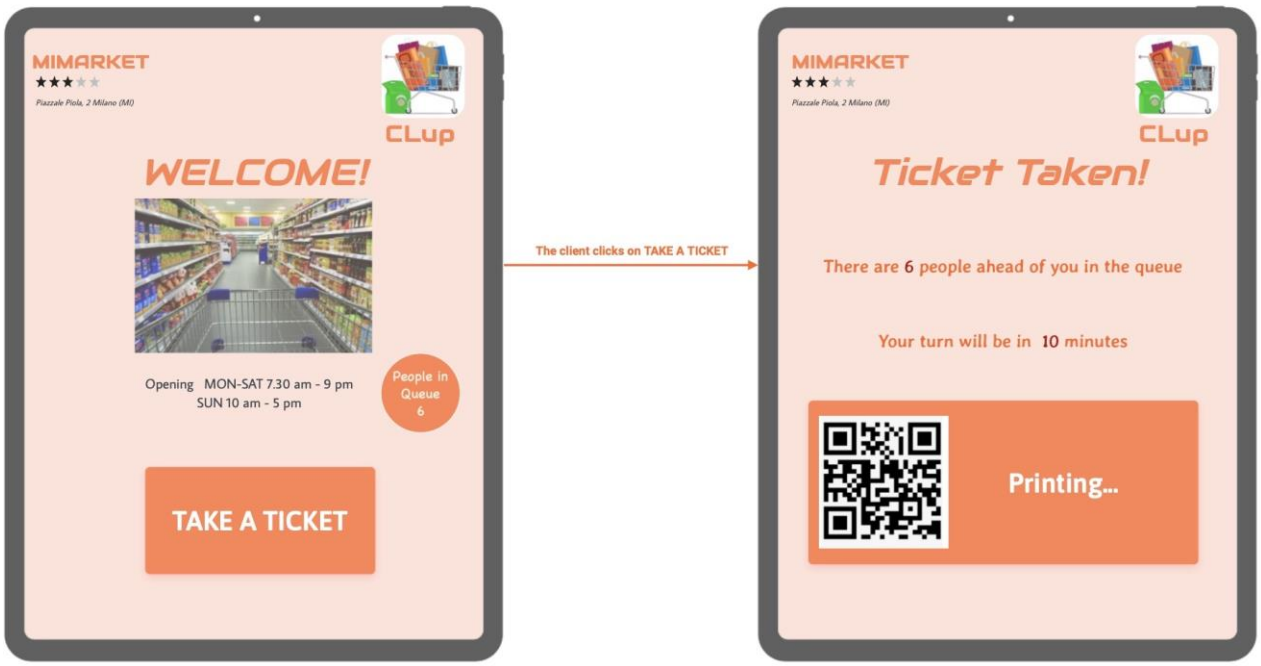




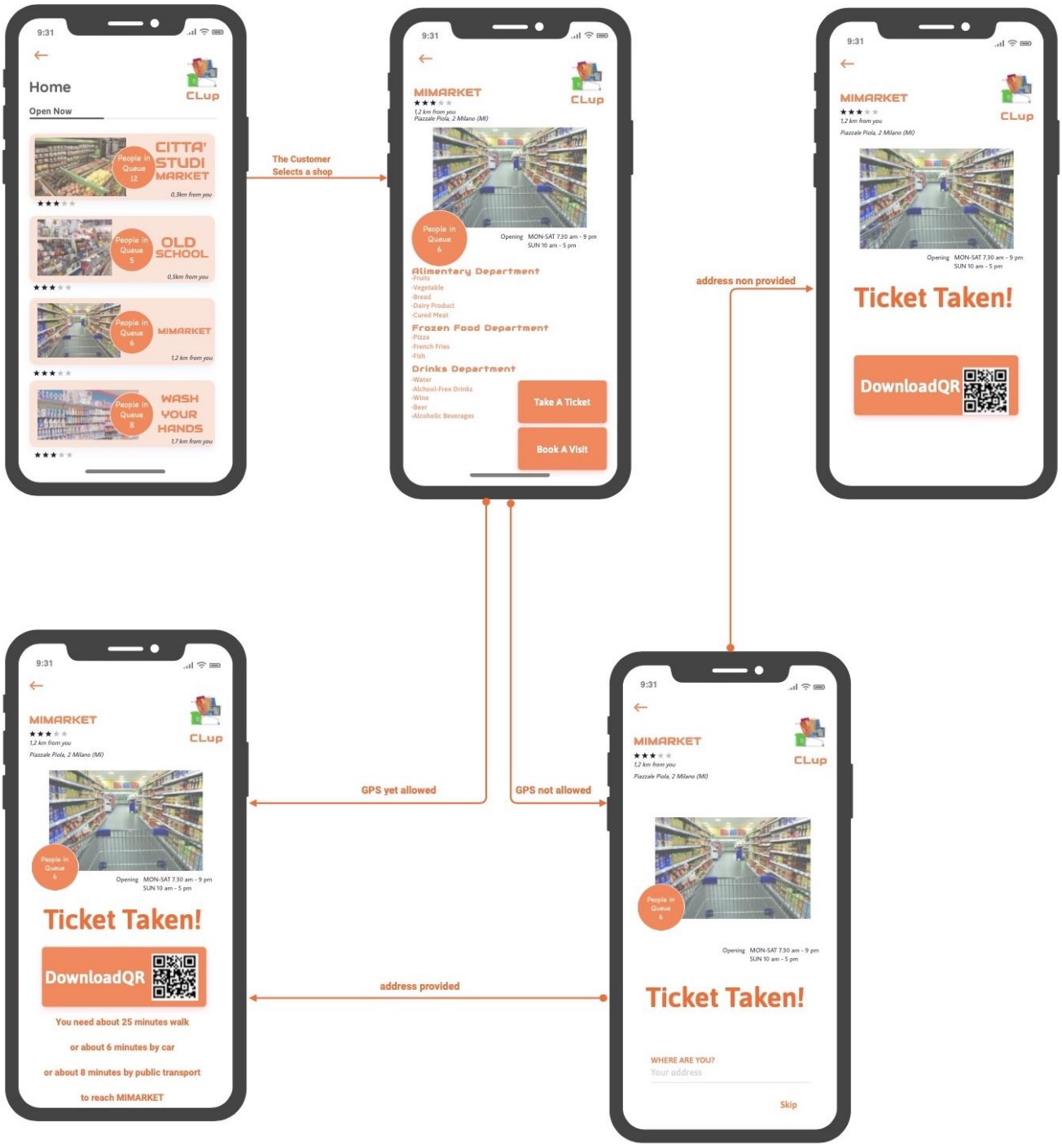
Login Customer and Shop Manager



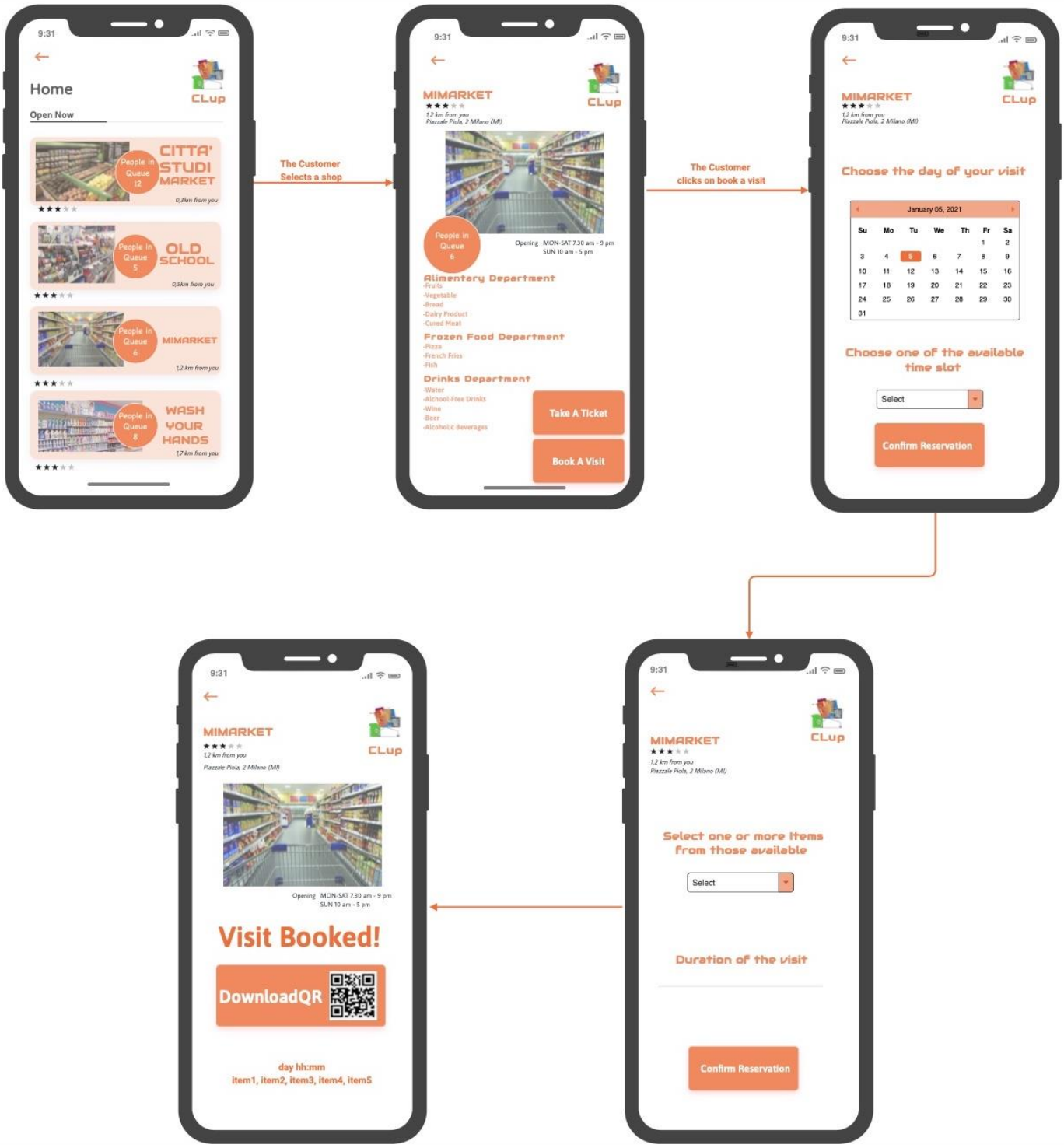
Take a Ticket on the Spot (Totem)



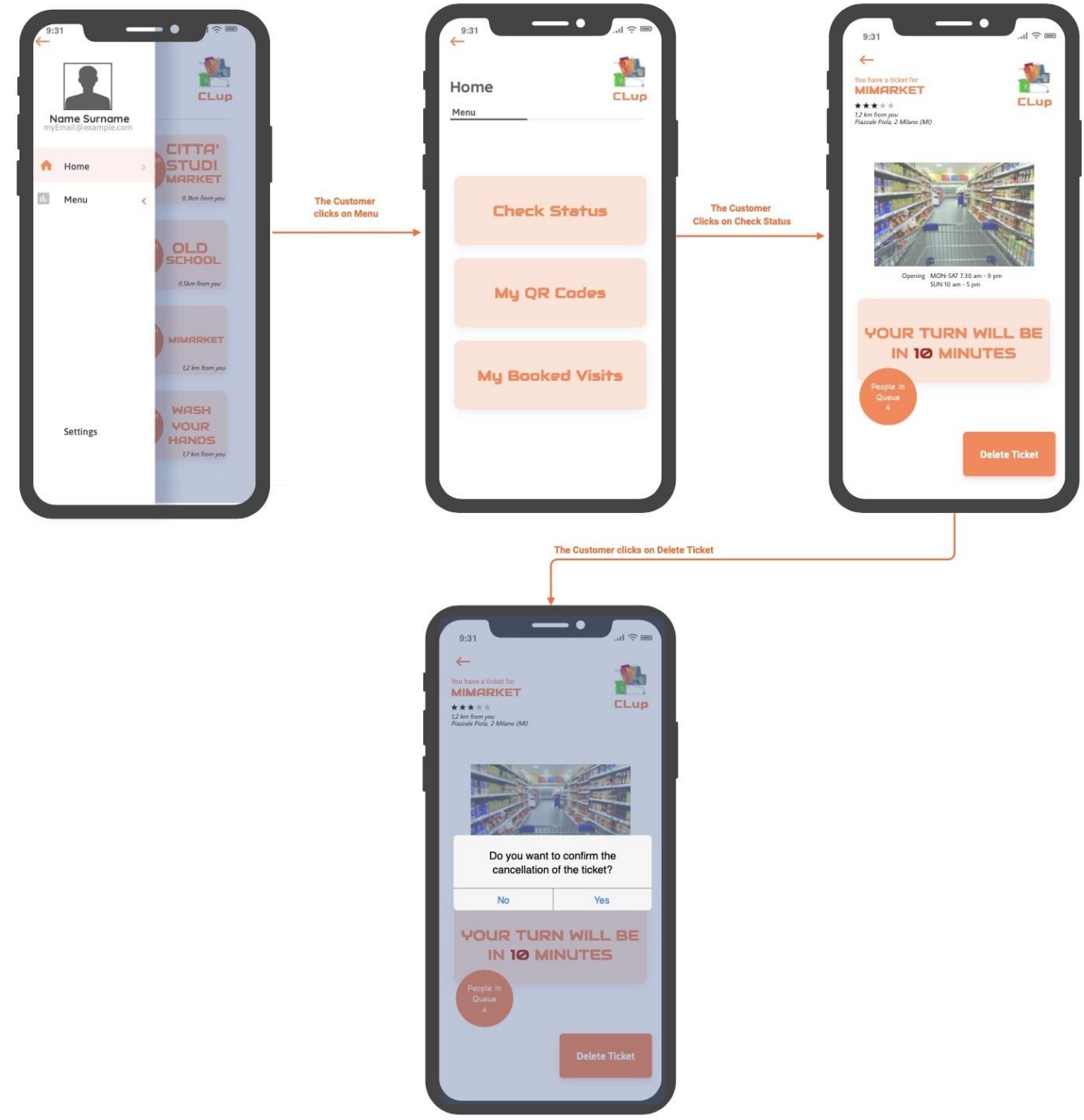
Take a Ticket Online (Customer)



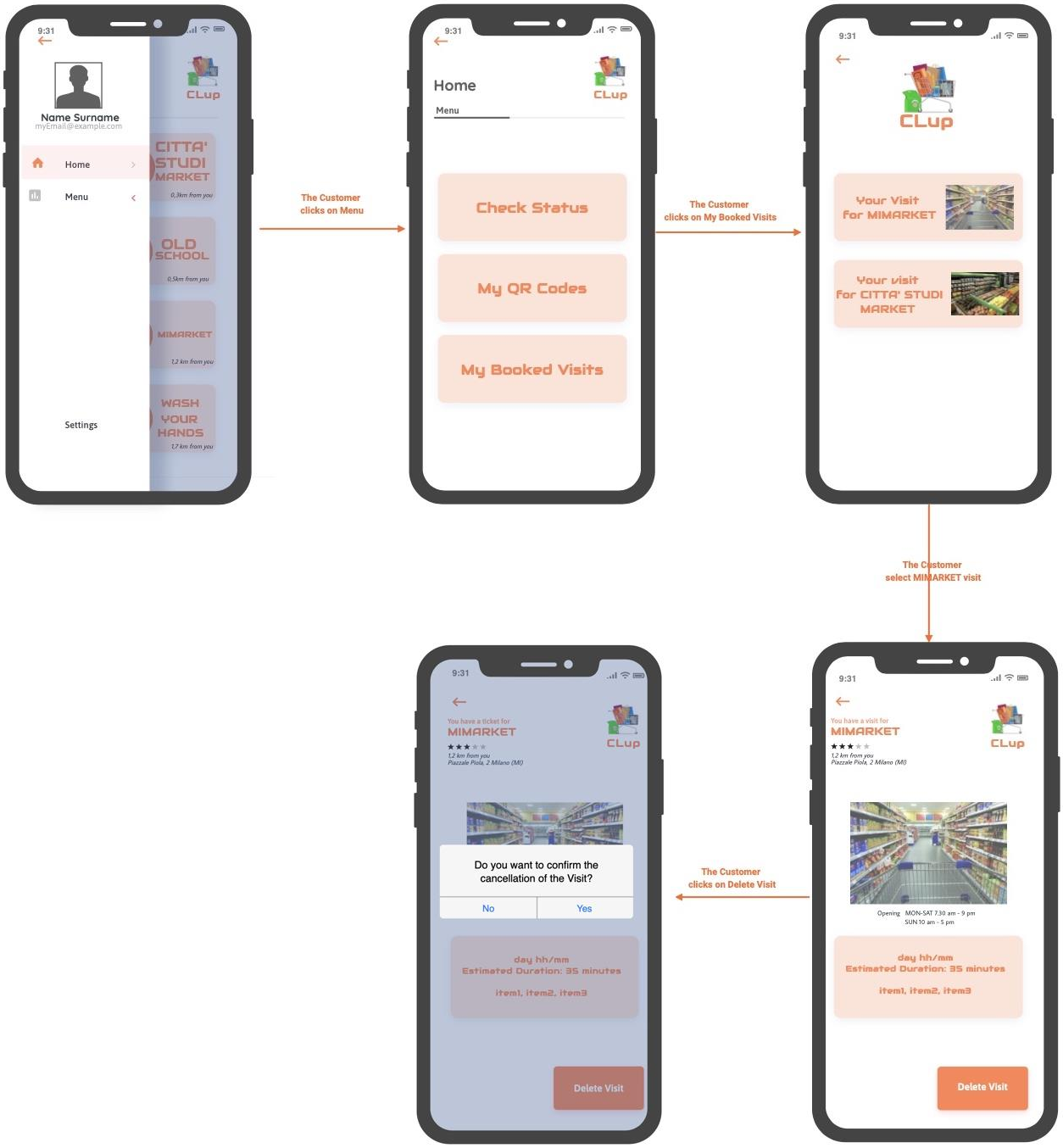
Book a Visit (Customer)



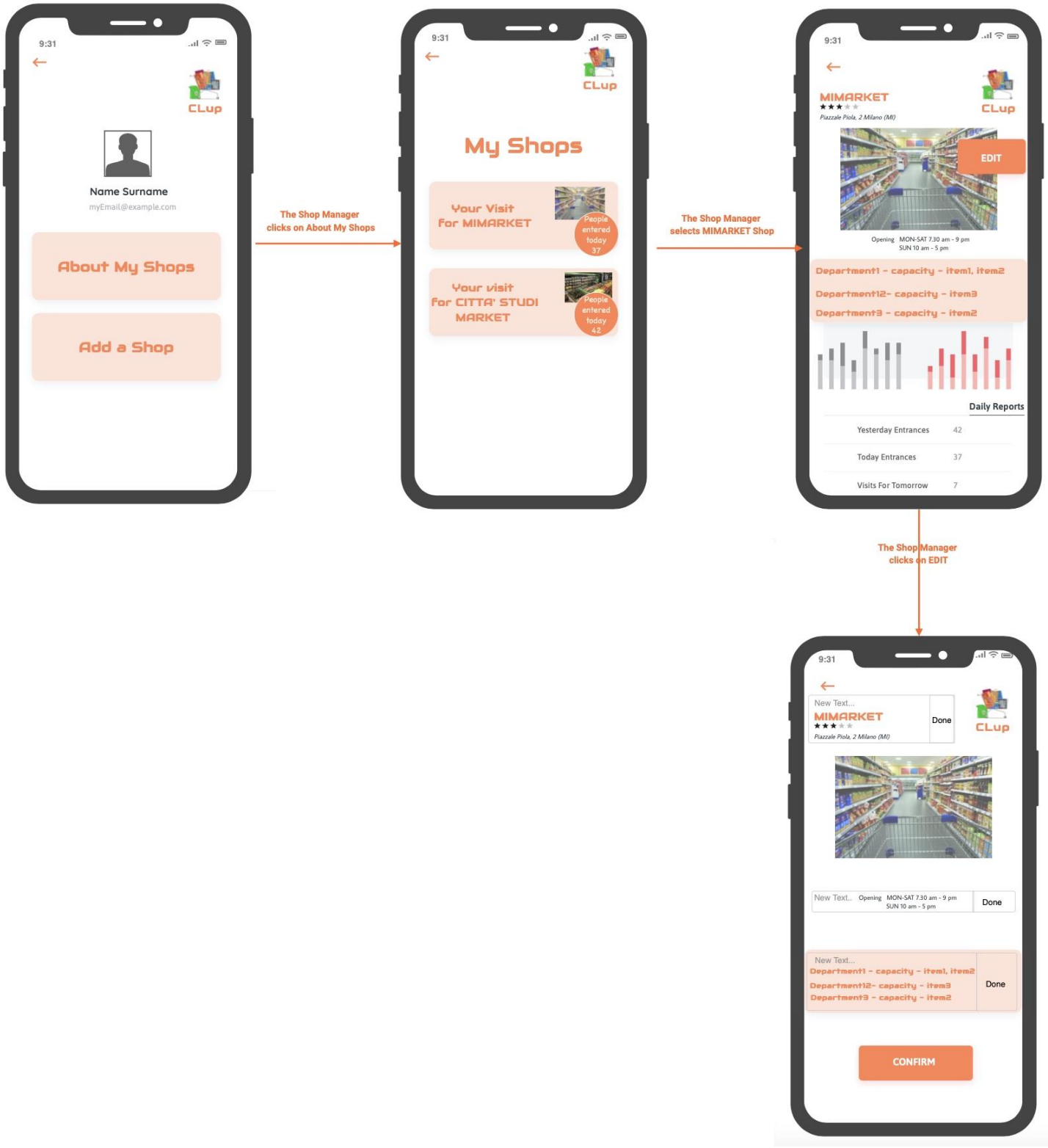
Check Status and Cancel Ticket (Customer)



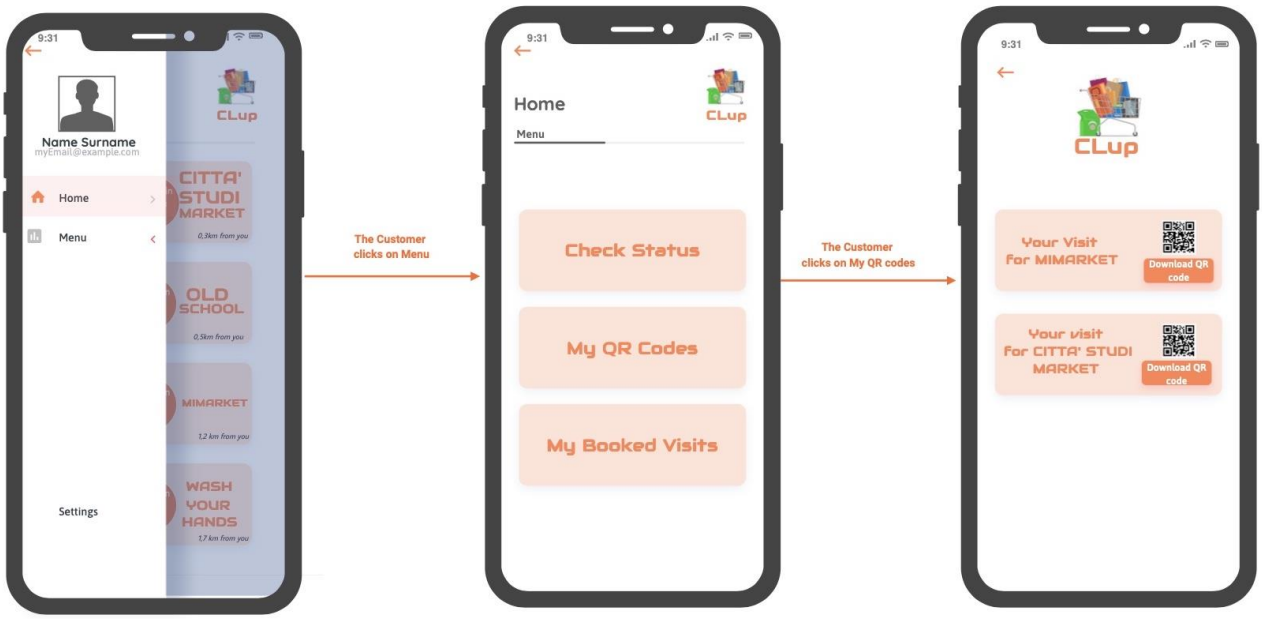
Check Booked Visits and Cancel Visit (Customer)



Check and Modify Shop's Information (Shop Manager)



Download QR Code (Customer)



4. Requirements Traceability

Goal	G1: ALLOW PEOPLE TO TAKE A TICKET ONLINE TO DO SHOPPING
Requirements	R2, R3, R4, R7, R10, R11, R13, R15, R19, R20, R22, R23, R24, R28
Components	GoogleMapsService UserCustomerMobileApp TakeTicketManager EnterAppManager (LoginService, RegistrationService, CheckPersonalInfoService*, QRsService*, ShopsInfoService*, QueueStatusService*) ScheduleManager DataManager (CancelTicketService, QueueStatusService) DBMS

Goal	G2: ALLOW PEOPLE TO TAKE THE TICKET ON THE SPOT
Requirements	R13, R14, R24
Components	UserCustomerMobileApp TotemManager TotemApp ScheduleManager DBMS

Goal	G3: ALLOW PEOPLE TO MAINTAIN DISTANCE RULES WHILE DOING SHOPPING
Requirements	R1, R7, R10, R11, R12, R13, R14, R16, R17, R20, R1, R22, R23, R24, R21
Components	UserCustomerMobileApp DataManager (MyShopsService) ScheduleManager RegistrationService TakeTicketManager DBMS BookVisitManager TotemManager TotemApp DataManager (QueueStatusService) GoogleMapsService GoogleCalendarService QRScannerService DBMS

Goal	G4: ALLOW STORE MANAGERS TO ORGANIZE IN A MORE EFFICIENT WAY THE STORE
Requirements	R1, R6, R11, R12, R18, R19, R21, R24, R25, R29
Components	UserShopManagerApp DataManager(MyShopsService, CheckPersonalInfoService*, AddShopService*, ModifyShopService*) EntranceManager (LoginService, RegistrationService) ScheduleManager GoogleCalendarService DBMS QRScannerService

Goal	G5: ALLOW PEOPLE TO AVOID LINES IN FRONT OF THE SHOP
Requirements	R1, R2, R7, R9, R11, R12, R14, R16, R17, R20, R22, R23, R21, R24
Components	UserCustomerMobileApp UserShopManagerMobileApp DataManager(MyShopsService, QueueStatusService,) GoogleMapsService ScheduleManager TakeTicketManager BookVisitManager TotemManager TotemApp GoogleCalendarService DBMS QRScannerService

Goal	G6: ALLOW STORE MANAGERS TO TRACK THE AFFLUENCE OF PEOPLE IN THEIR SHOPS
Requirements	R1, R6, R13, R29
Components	UserShopManagerMobileApp DataManager (MyShopsService, AddShopService*, ModifyShopService*) EnterAppManager (LoginService, RegistrationService) TakeTicketManager TotemManager TotemApp BookVisitManager GoogleCalendarService DBMS QRScannerService

Goal	G7: ALLOW PEOPLE TO BOOK A VISIT ONLINE TO DO SHOPPING
Requirements	R3, R5, R27, R8, R9, R10, R12, R13, R15, R16, R17, R18, R21, R28
Components	UserCustomerMobileApp BookVisitManager EnterAppManager (LoginService) ScheduleManager DataManager(CancelVisit, CheckPersonalInfoService*, BookedVisitsService*, QRsService*, ShopsInfoService*) DBMS GoogleCalendarService DBMS

Goal	G8: ALLOW THE SYSTEM TO BUILD STATISTICS ON DATA WHICH ARE STORED IN THE DATABASE
Requirements	R1, R10, R8, R11, R12, R14, R16, R17, R24, R25, R21, R28, R29
Components	UserCustomerMobileApp UserShopManagerMobileApp DataManager (MyShopsService) DBMS ScheduleManager GoogleCalendarService DBMS QRScannerService

The services with * are additional and optional functionalities that a user can exploit in any moment. For this reason, they are not strictly necessary for the related goal

R1. The system should provide the report of daily entrances/exits to the shop managers
R2. The system should ask the user-customer for authorization of GPS position
R3. The system should ask the user to tick privacy Terms of Services and Privacy Policy box
R4. The system should require the user- customer to be logged in to exploit the services related to Take a Ticket option provided by the application
R5. The system should require the user- customer to be logged in to exploit the services related to Book a visit option provided by the application
R6. The system should require the user-shop manager to be logged to exploit the services provided by the application
R7. The system should notify the user- customer (about the remaining waiting time) real time—ScheduleManager
R8. The system should ask the user- customer who want to book a visit to fill the mandatory fields of date and time of the visit
R9. The system notifies the user- customer two hours before the booked visit to remind it to him/her
R10. The system must save the data provided by the users when they register themselves
R11. The system must save the generated ticket taken online in the schedule
R12. The system should save the ticket of Book a visit real time in the schedule
R13. The system should generate automatically the QR code associated to the ticket
R14. The system should save real time the ticket taken through the totem in the schedule
R15. The system should allow the user-customer to decide between Take a ticket and book a visit option
R16. The system should allow the user-customer to provide in an optional field the category of items they want to buy (in Book a visit option)
R17. The system should allow the user-customer to provide in an optional field the estimated duration of the visit (in Book a visit option)
R18. The system should allow the user-customer to cancel the reservation of a visit until one hour before the starting of the visit
R19. The system should allow the user-customer to cancel a ticket within his/her turn
R20. The system should allow the user-customer to check the status of a queue of a shop
R21.The system should reserve a place in the selected time slot for each customer who books a visit
R22. The system should allow the user-customer to check in each moment the remaining time for his/her turn
R23. The system should allow the user-customer to check in each moment the number of people before him/her in a queue of a shop
R24. The system should assign a position in the queue for each customer who takes a ticket
R25. The system should allow the user-shop manager to edit information about their shop
R26. The system should ask the store manager to provide a certificate that validates his/her role
R27. The system should require the user- customer to be logged in to exploit the secondary services provided by book a visit (see information of the user, edit information…)
R28. The system should ask the user-customer to register himself/herself to the application by filling the form with mandatory fields
R29. The system should ask the user-store manager to register himself/herself to the application by filling the form with mandatory fields

5. Implementation, Integration and Test Plan

5.1 Implementation Plan

The approach used to implement the components of the system is the **Bottom-up** one with which the decision-making and process implementation comes from the lower levels and proceeds upwards.

Indeed, by using the Bottom-Up approach, the base elements of the system are first specified in great detail and then linked together to create larger subsystems until a complete top-level one is formed. In this way the final system is created in an incremental way and this allows the testing in parallel with the implementation.

Starting testing in early phases can bring several benefits since it facilitates the risk identification: it gives the possibility to find out bugs and any other kind of problems from the beginning. This allows to fix them with less effort than if the system were nearly complete.
The decision also comes from the fact that with this approach it is possible to implement password management for a large number of users.

The subsystems are implemented starting from the components on which the other ones rely on. As shown in the table provided below, the order is also defined based on the difficulty of the implementation of the components and on their importance.

COMPONENT	FEATURE	DIFFICULTY	IMPORTANCE
DBMS	Data Storing	Highest	Highest
EnterAppManager	User login and signup	Low	Low
TakeTicketManager	User customer “Take a Ticket” online option	High	High
BookVisitManager	User customer “Book a Visit” option	High	High
TotemManager	User-customer “Take a ticket on the spot” option	High	High
ScheduleManager	Managing the daily schedules of each shop in the database. It is also used for the generation of the daily reports for the user shop manage	Medium	Medium
DataManager	Provides additional secondary functionalities for the users (like checking the queue, modify/add/check some stored data)	Medium	Low

Since the development of the **DBMS** is the most important one, it represents the first step of the implementation plan. Indeed, it is thanks to the DBMS that is possible to have access to the database to get and store the information. Every time a data is needed it is necessary to query the database and retrieve what is required. So, all the methods used to add, update, remove tuples from/to the database are provided by this component.

The second step regards the TakeTicketManager, BookVisitManager and TotemManager, which correspond to "high" status in the table. To develop them it is necessary to first develop the **ScheduleManager** (despite the "medium" difficulty and importance), which deals with all the functionalities related to daily reservations of each shop. About the "*Take a ticket"* and the *"Take a ticket on the spot*" options the ScheduleManager is necessary to see the status of the queue, check which position can be assigned to the user and decide whether to accept or not a ticket request.
Only in the case of "Take a Ticket" the interaction with the ScheduleManager is also necessary to compute the number of people above him/her in the queue. Regarding the *"Book a visit"* option the Schedule Manager gives to the BookVisitManager the free slots of the selected day and interacts with it to make it decide whether to refuse or not the reservation.

Then the **TakeTicketManager**, **BookVisitManager** and **TotemManager** can be implemented with all their functionalities. The TicketManager deals with the tickets requests that the customers do online and for this purpose it interacts both with the Schedule Manager and the DBMS. The TotemManager has the same functionality but deals with the people who want to take a ticket on the spot. Regarding the BookVisitManager, it manages the requests of visit's reservations made by the customer. As in the previous case it needs to interact with the ScheduleManager and the DBMS to store and get all the information needed.

Despite both DataManager and EnterAppManager having the same "Low" status, the following step is characterized by the implementation of **DataManager** since it has "medium" difficulty. Moreover, its AddShopService is needed to develop the Shop-manager SignUpService of the EnterAppManager.
The other services of DataManager are: CheckPersonalInfoService, QueueStatusService, BookedVisitsService, QRsService, ShopsInfoService, MyShopsService, CancelTicketService, ModifyShopService. They can be implemented without any particular order since the majority of them are independent one from each other and rely only on components that will have already been implemented at this point.

Then follows the implementation of **EnterAppManager,** with the RegistrationService and LoginService. It has a low difficulty and regards the operations of Login and Registration both of the User-customer and user shop manager.

The last step regards the development of the **Router**, which is the component responsible for forwarding the messages from the client part to the server one and vice versa. It is implemented only at the end because, although it is one of the most fundamental components, it does not contain too much business logic.

Regarding the external services GoogleMapsService, GoogleCalendarService, QRScannerService, they are provided and so there is no need to implement them (and also do unit tests). They will be only tested during the integration phase.
To summarize, the order of implementation that must be followed is:

1. DBMS
2. ScheduleManager
3. BookVisitManager, TakeTicketManager, TotemManager
4. DataManager
5. EnterAppManager
6. Router

Instead, UserCustomerMobileApp, UserShopManagerMobileApp, TotemApp can be implemented in parallel to the implementation of all the other components shown above.

5.2 Integration Strategy

With respect to the Integration and Testing phase, unit testing starts with the module specification and can be performed at any time, while the integration testing is done after the unit testing and before the system testing. Its aim is to detect any irregularity between the units that are integrated together.

As in the case of the implementation strategy, also the integration one is based on a **Bottom-Up** approach: an initial incomplete version is constructed and then progressively more components are added. So low-level modules are tested first, and then high-level modules are tested by following an incremental approach.

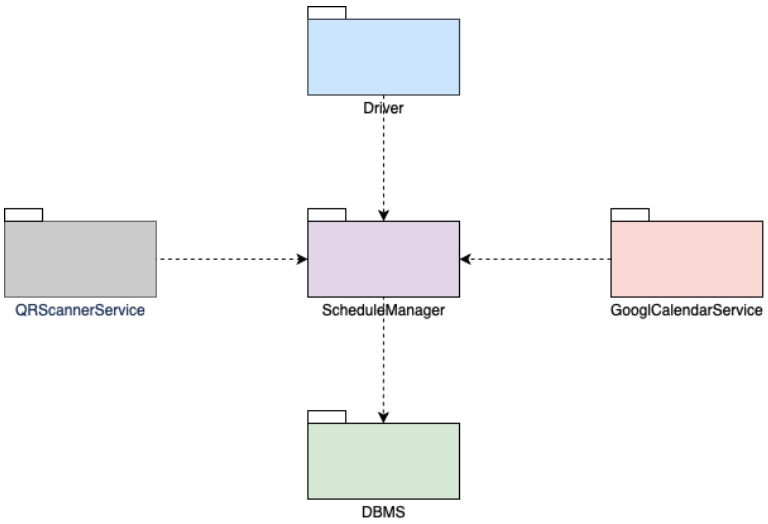
This decision comes from the fact that:

- the Bottom-Up approach is used also for the Implementation strategy
- it is easier to create test conditions and observe the results

For this strategy Drivers will be used: they are pieces of software that invoke the Unit being tested by creating necessary 'Inputs' required for the Unit. They keep a minimum level of complexity, so that they do not induce any errors while testing the unit in question.
In the following drawings the arrows start from the component that 'uses' the other ones.

Server-side components

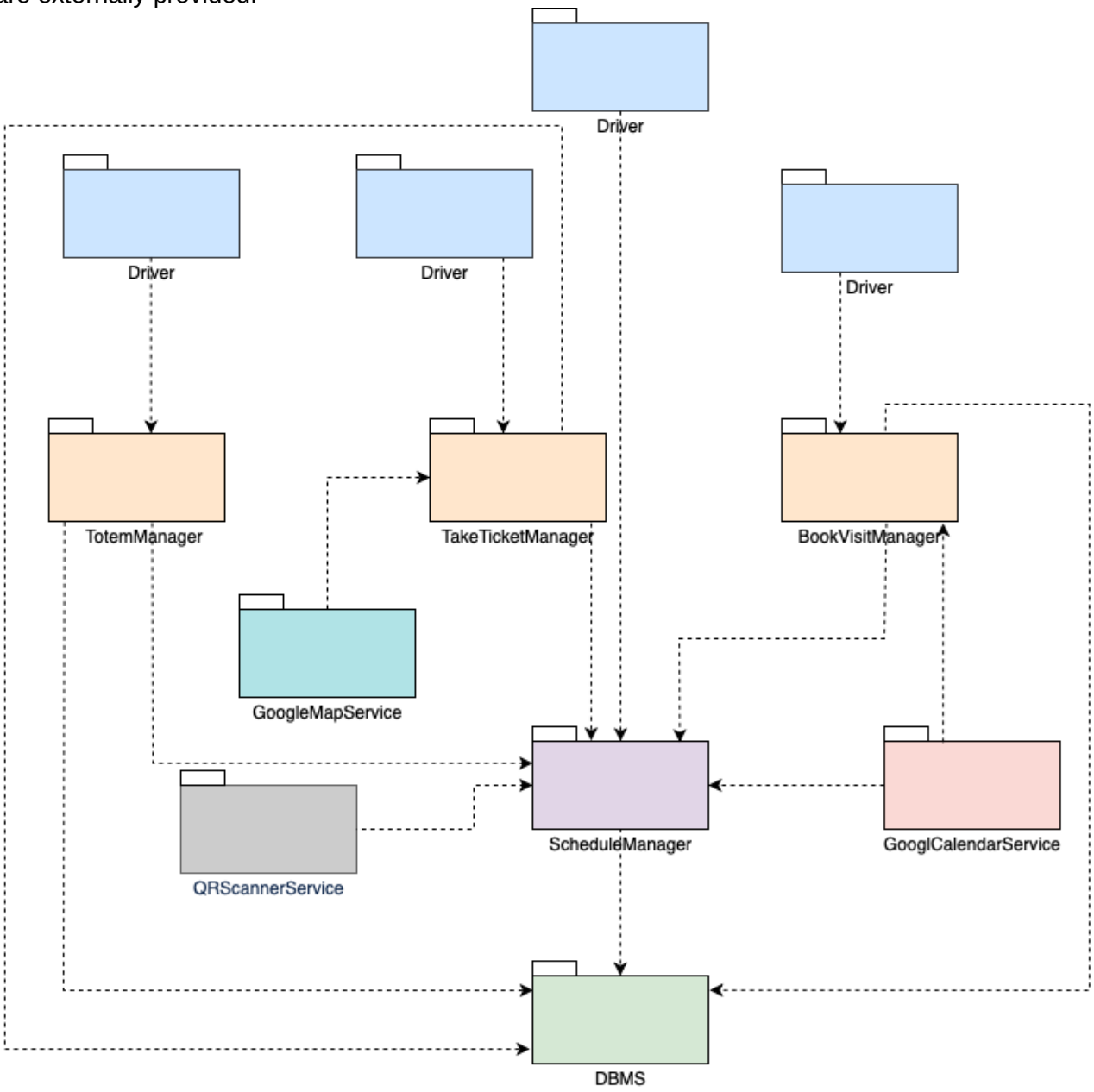
Specifically, the first component that is implemented and unit tested is the **ScheduleManage**r, which is integrated with the **DBMS** that has been already implemented. It is the first one since - as previously mentioned - all the other components rely on it. The ScheduleManager must also be integrated with GoogleCalendarService and QRScannerService (to receive the real time of customers' entrances/exits associated with each scanned QR code), which does not need to be implemented.



The second step regards the implementation and unit testing of the three components: **TakeTicketManager**, **BookVisitManager** and **TotemManager**. They represent the main functionalities of CLup Application and since they can be considered independent one from the other, they can be unit tested in parallel.

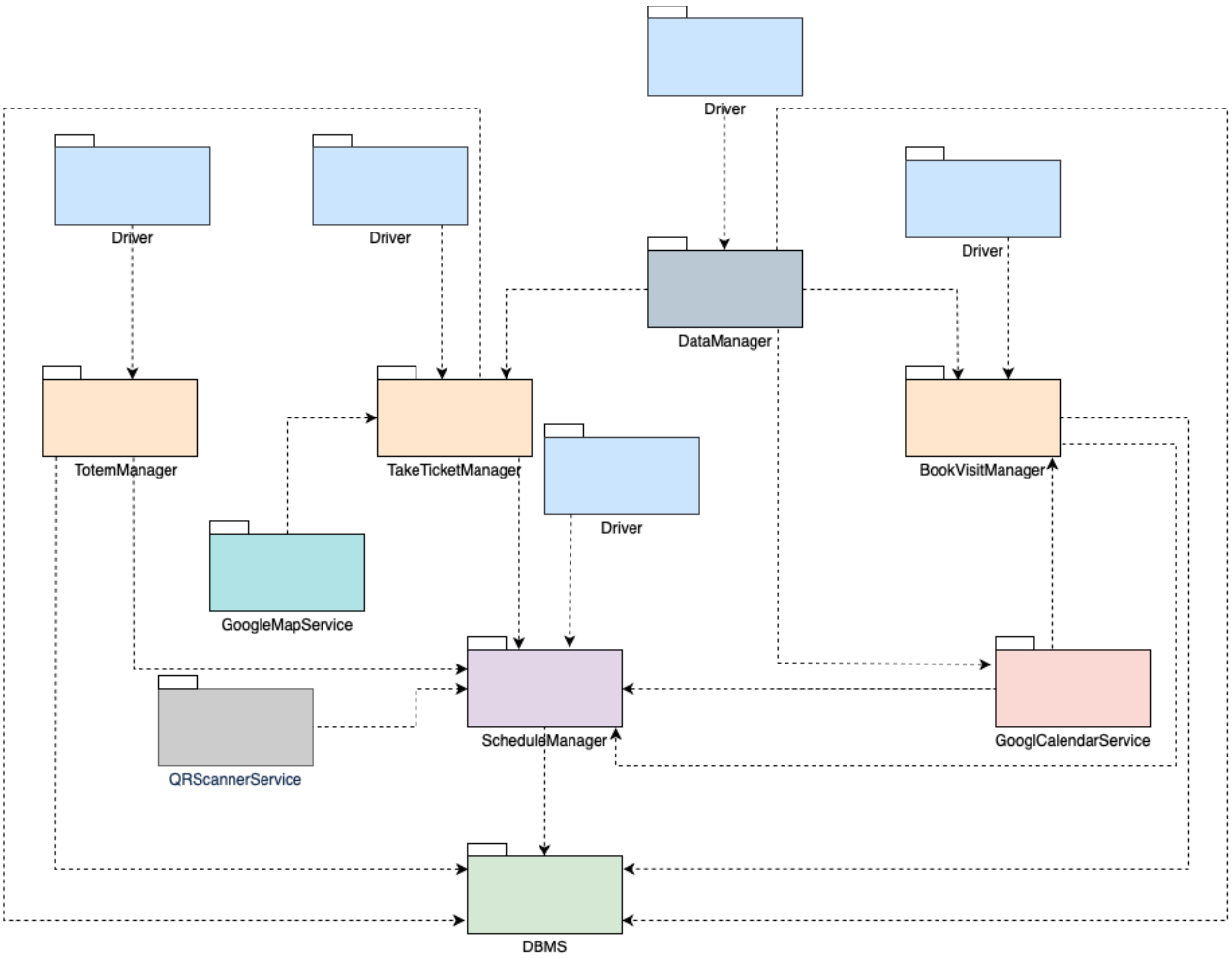
Once all of them are correctly tested, the integration with the ScheduleManager can start. They need this component since the mentioned managers (ticket, visit, totem) cannot accept a reservation without interacting with the ScheduleManager which gets the necessary data from the DBMS. Also the two managers are connected to the database to save the QR code after generating it.

TakeTicketManager must be integrated also with the GoogleMapsService and the BookVisitManager with the GoogleCalendarService. They do not need to be unit tested since they are externally provided.



In the third step **DataManager** is implemented, unit tested and integrated. Only BookVisitManager and TakeTicketManager rely on it and so it must be integrated only with them. TotemManager does not need the secondary functionalities provided by DataManager.

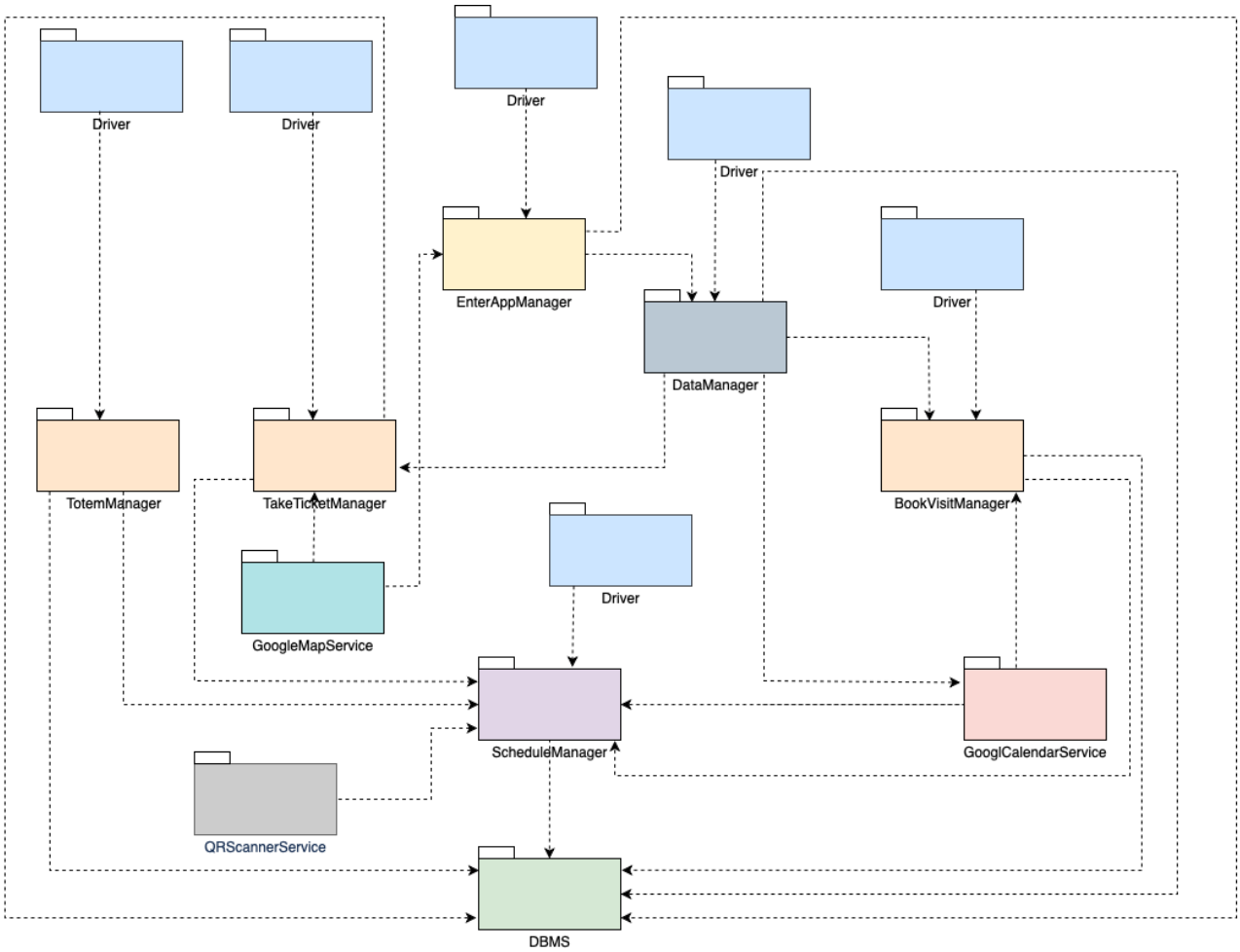
DataManager is also connected to the DBMS, from which it gets the data it needs - and to Google Calendar.



The following step regards the last component, the **EnterAppManager.**

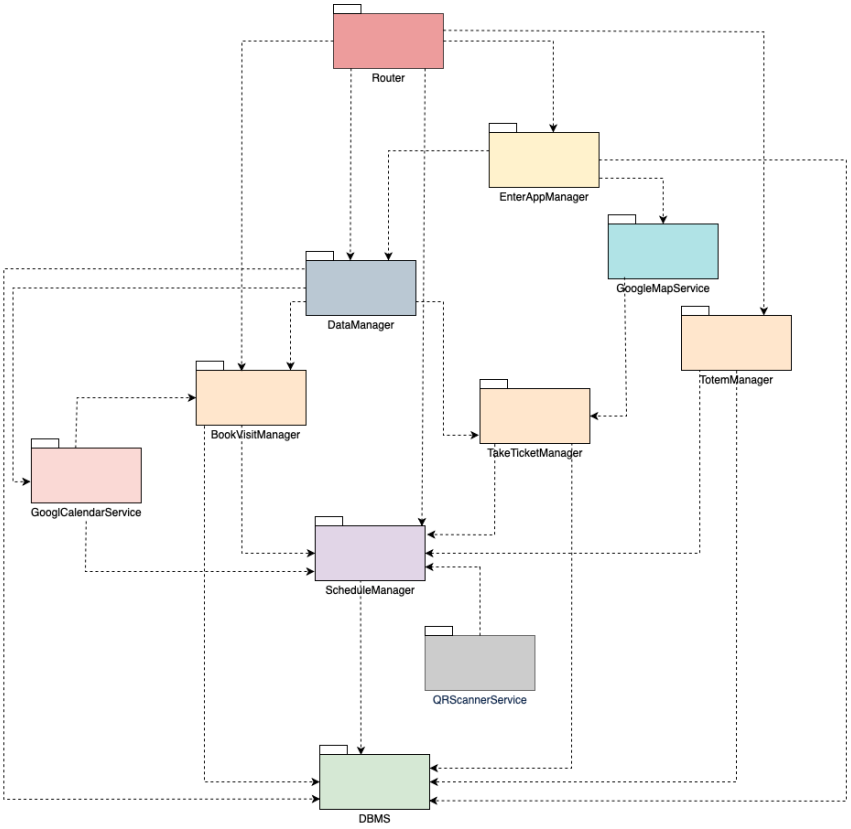
Since it needs the DataManager (specifically the AddShopService) only for the RegistrationService, the other Service for the Login can be unit tested in parallel with all the other components. Instead, the RegistrationService must wait for the AddShopService' s unit test.

Only when both the Registration and Login Services have been successfully unit tested, the integration with the DataManager can be done.
In this step also the GoogleMapsService is integrated with the EnterAppManager.



The last step regards the substitution of the drivers with the **Router**, which has to be implemented, unit tested and finally integrated with all the components (except for the external services).

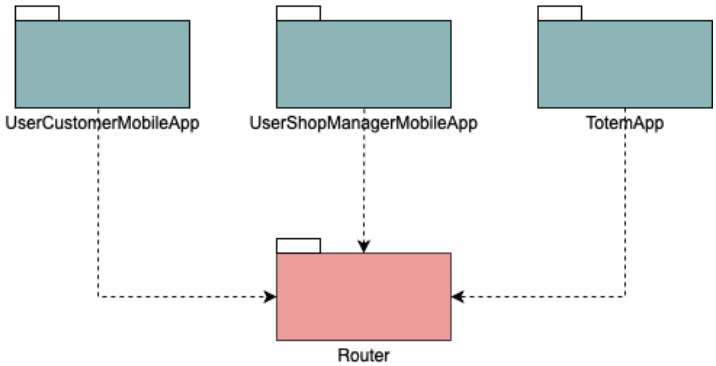
It is connected to all of them since it is the dispatcher of the requests.



Client-Side Components

The integration of the client-side components can be done once all the ones of the server side have been successfully integrated. Instead - as explained in the previous section - their implementation and unit testing can be done in parallel with the components of the server side.

Specifically, all of them (UserCustomerMobileApp, UserShopManagerMobileApp, TotemApp) must be connected to the Router.



5.3 System Testing

As anticipated in the previous section, the System Testing can be done once the whole integration has been successfully completed. In fact, it must be performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements (functional and non-functional) expressed in the RASD document. It aims to detect defects within both the integrated units and the whole system.

The system tests to be performed are:

- **Load Testing:** it understands the behavior of the application under a specific expected load. For example, it is used to ensure that the application performs satisfactorily when multiple users try to access or use it at the same time.

- **Stress Testing:** it measures the robustness of the system under extremely heavy load conditions and it is carried out until the system failure. It is used to determine the limit at which the software breaks and to demonstrate effective error management under extreme conditions.

- **Performance Testing:** it is carried out to test the speed, scalability, stability and reliability of the software. It must ensure that the limits imposed in the Performance Requirements are satisfied according to what has been written in the RASD Document. Specifically:
  - the response time for any action must be less or equal to 1 second.
  - the system must generate the QR code within 3 seconds from the “take a ticket” and “Book a Visit” request.
  - run the algorithms implemented to manage the data in the schedule within two seconds from the action of the user.

- **Scalability Testing:** ensures the capability of software to scale up with the increasing workload and data traffic and since CLup in the future could involve more and more users, it creases the confidence to meet the future growth demands.

- **Functionality Testing:** it performs to evaluate if the system satisfies the functional requirements expressed in the RASD document.

- **Security Testing:** it measures the potential vulnerabilities of the system by detecting each possible security risk in the system. It ensures that the application is free from any threats or risks that can cause data loss.

6. Effort Spent

We have done about 70% of the work together in videocall, especially for the decision making. We have divided the remaining part equally.
We have also used a Google Doc to fix some aspects autonomously and in parallel.

Preliminary discussion	1h
Final reading	4h

Topics	Hours
Purpose and Scope	Member1: 1h Member2: 1h
Definitions, Acronyms, Abbreviations, Reference Documents, Document structure	Member 1: 1h Member 2: 1h
Overview	Member 1: 2h Member 2: 2h
Component view	Member1: 3h Member2: 4h
Deployment View	Member1: 3h Member2: 1h
Runtime View	Member1: 6h Member2: 6h
Selected architectural styles and patterns	Member1: 2h30 Member2: 2h
Other design decisions	Member1: 1h Member2: 1h30
User Interface Design	Member 1: 3h Member2: 4h
Requirements Traceability	Member 1: 2h Member2: 2h
Implementation Plan	Member1: 3h Member2: 2h
Integration Plan and System Testing	Member1: 4h Member2: 4h

Member 1: Alessia De Berardinis
Member 2: Arianna Dragoni