DD

Design Document

POLITECNICO
MILANO 1863

Paolini Giuseppe
Matricola:968534

Pellini Riccardo
Matricola:968866

Panzeri Matteo
Matricola:965568

A.Y. 2020-2021

Professore:
Rossi Matteo

Index

1. Introduction

1.1 Purpose

The design document aims to give a technical description of the project introduced in the RASD.

The document shows how the project has to be developed both from the hardware and the software point of view.

The DD explains the architecture of the system and the software components used by it.

At the end of the document is present the implementation, integration and test plan.

1.2 Scope

Due the Covid-19 pandemic is extremely important avoid crowds in public places.
In particular this project aims to regulate the number of people that access a supermarket via a system which is based on reservations which are made using a smartphone or through a supermarket employee.

A reservation generates a QRCode which is used to access the supermarket.

If the reservation is made using the application a report is created. It contais the details of the user's visit to the supermarket.
For habitual customers the system can generate automatic suggestions.

The project is designed in order to grant the observation of the social distancing rules while maximizing the number of people in the store.

1.3 Definitions, acronyms and abbreviations.

1.3.1 Definitions

Temporal slot	Amount of time in which a customer can access the supermarket. Defined by a day and a starting hour.
Ticket	Receipt sent to the user after a line up or a reservation it contains the QR code.
Unit	Defined section of the supermarket which contains the same categories of goods as pasta, vegetables and fruit, meet etc.
Available spots	Given an unit the Available spots are the number of people who can stay in that unit at the same time given a day and an hour.
Store employee	Employee of a supermarket who will act as a proxy between who wants to reserve a slot without using the application.

Line up	Process via an user reserve the first available slot.
Reservation	Process via an user selects the temporal slots given a list of them
Utility desk	Physical place outside the supermarket where people can go in order to reserve a slot if they cannot use the application.
QR code	Code in the ticket which is used to access and go out a supermarket it is unique and related to a single reservation.
QR reader	Device placed at every entrance and exit of a supermarket it is used to detect and read the QR codes of the users.
Time of visit	Time that the user plans to spend in the supermarket if it is not specified it will be assumed as an hour.
Notification	Message sent by the system to the user when the system thinks the user should leave his address in order to reach the supermarket on time.

1.3.2 Acronyms

GPS	Global Positioning System

1.3.3 Abbreviations

Slot	Temporal Slot
Reservation	Sometimes it is also used for the line up
WPn	World phenomena number n
Gn	Goal number n
SPn	Shared phenomena number n
Dn	Domani assumption number n

1.4 Revision History

1.5 Reference Document

This DD document is based on the
Requirement Engineering and Design Project: goal, schedule, and rules document given by the professor.

1.6 Document structure

- Chapter 1 describe the document purpose and structure, it also describe the document content.
- Chapter 2 describe the architectural design of the project. It contains the components, interfaces definitions and the description of the interaction among components.
- Chapter 3 shows with mockups how the user interface should be developed.
- Chapter 4 is the bridge between the RASD and this document. It maps which components are used in order to reach the goals defined in the RASD.
- Chapter 5 describe the implementation, integration and testing plan. This is done in order to develop the project in an efficient way without any unnecessary time loss.
- Chapter 6 shows the effort spent by each component of the group.
- Chapter 7 include the reference documents.

2. Architectural Design

2.1 Overview

The application is developed using the client-server paradigm.

In our project the client can be the users' devices or the employee PC they communicate with the server of the application which handles the requests and store the information on the distributed DB (every supermarket will have his DB).

We can divide the application architecture in three levels:

1. Presentation Level: this collects the requests of the users sends the requests and manage the answers.
2. Business logic level: This is the core of the application in fact it handles the requests.
3. Data access level: This level manages the data storage.

2.1.1 High level components

The application will be distributed among 4 tiers. The three levels described in precedence will be distributed among these four tiers.

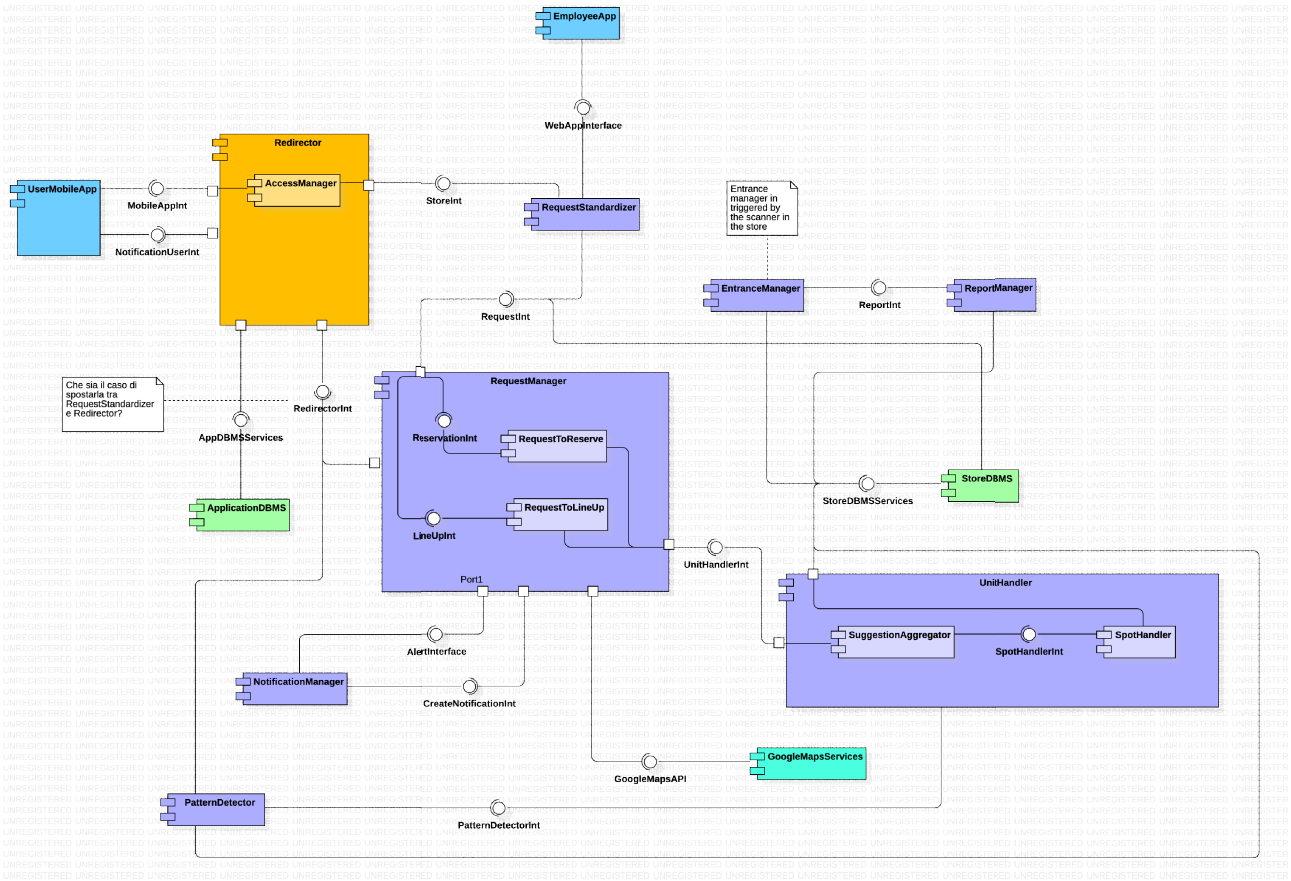
A tier will implement the presentation level.

A tier will be used by the data access level.
The business logic level will be divided between two different tiers one will collect the requests and send them to the right server (the one located in the desiderated supermarket) the second tier will be handle the requests received from the first one.

In the client node there will be only the presentation layer following the thin client model.

2.2 Component View

The following diagram illustrate the software components of the application server.

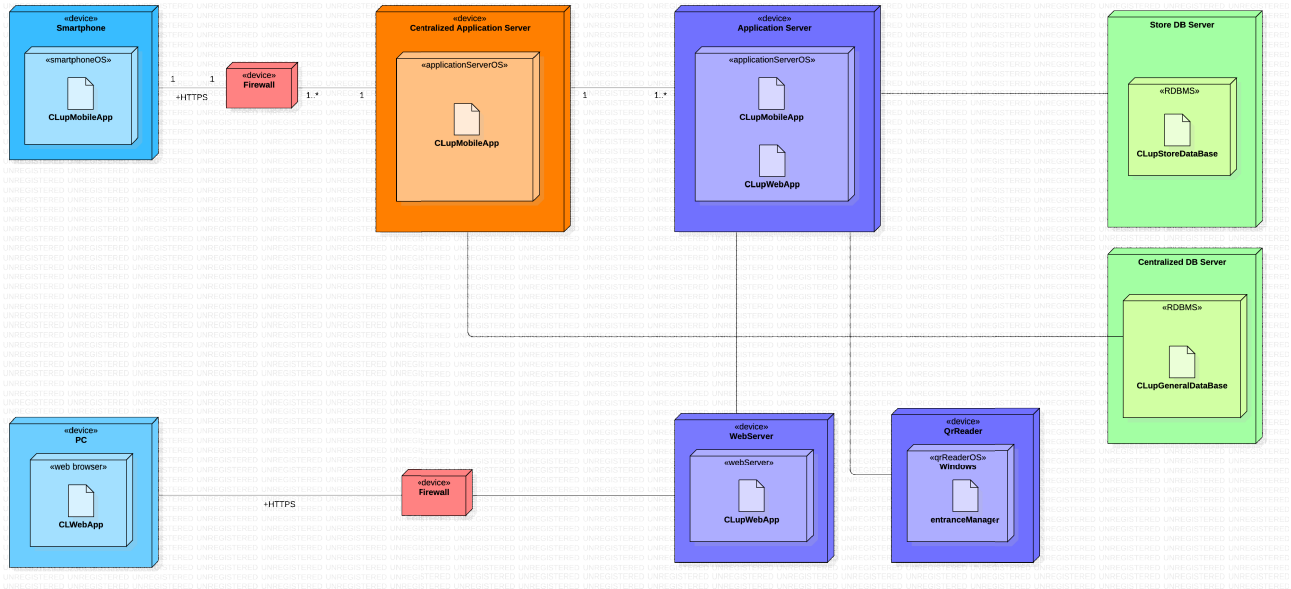


The main components are:

- Redirector: It is the component which collects the requests of the clients (sent via the application).
  If they are login or registration requests they are handled by the AccessManager otherwise they are submitted to the RequestStandardizer.
    - AccessManager: It is a subcomponent of Redirector and handles the users logins and registration attempts.
- RequestStandardizer: It standardizes the requests that came from the application and the employee PC in this way the RequestManager handles only one request format.
- RequestManager: The RequestManager is the component which makes the reservations and handles the notification sending.
    - RequestToReserve: It is the subcomponent which handles the reservation requests.
    - RequestToLineUp: It is the subcomponent which handles the lineup requests.

- NotificationManager: The notification manager is the component that creates and triggers the notification which are sent to the user.
- PatternDetector: It is the component which handles the reservations made through pattern detection.
- UnitHandler: It is the component which controls the unit's slot availability
    - SuggestionAggregator: it is a subcomponent which collects the slots and available spots and creates the suggestions which are sent to the user.
    - SpotHandler: it is the subcomponet which increase and decrease the spot availability of the units.
- EntranceManager: The entrance manager is the component which manages the QRCode readers and the access to the supermarket.
- ReportManager: The report manager is the component that creates the reports after a customer has left the supermarket.
- GoogleMapServices: It is an external component and it is used to compute the travel time of the customer from his/her address to the supermarket.

2.3 Deployment View



The main components of the deployment diagram are:

- Smartphone: The smartphone is the device used by the customer to reserve a slot. It communicates with the system via an HTTPS connection.
- PC: The PC is used by the employee at the utility desk.
- Centralized Application Server: The centralized application server is the server which receive the requests, redirects them and manage the login and registration of the users.

It also allows the user to formalize her/his requests giving the store and units availability.

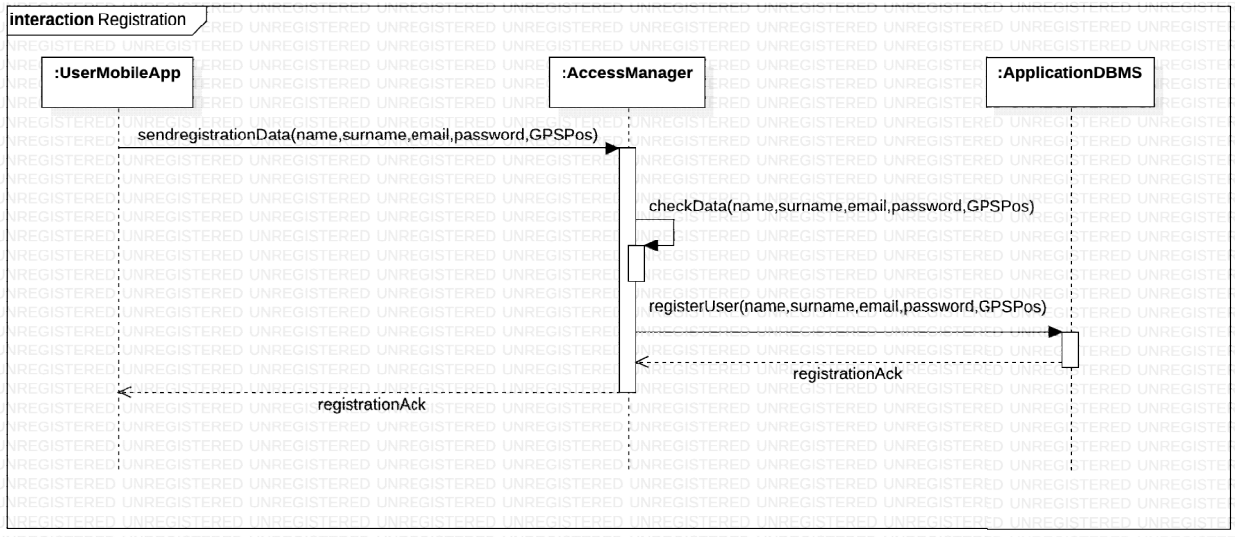
- Application Server: The application Server manage the requests standardized and submitted by the centralized application server and it the component which manages the spot and slot availability and compose reports and notifications.
- Web Server: the web server is the components which handles the requests sent by the employee pc.
- QRReaders: The QRReaders are the devices placed outside every entrance and exit of the supermarket they are connected to a PC which when a QRCode Is detected decide whether the customer is allowed to access the store.
- Centralized DB Server It is the data base which contains the users data like username and password and contains also the data related to stores and available units in them.
- Store DB Server: It is a part of the db which contains the reports data there are a part of the db in every store.(The reports of a store remains in the store)

2.4 Runtime View

2.4.1 Registration

The registration process begins with the user that (via the mobileApp) sends the required data. The access manager checks if the sent data are complete and correct and if they are the Access manager saves the new user's data in the DB using the ApplicationDBMS component.

The registration is completed when the mobile app receives the confirmation by the access manager.

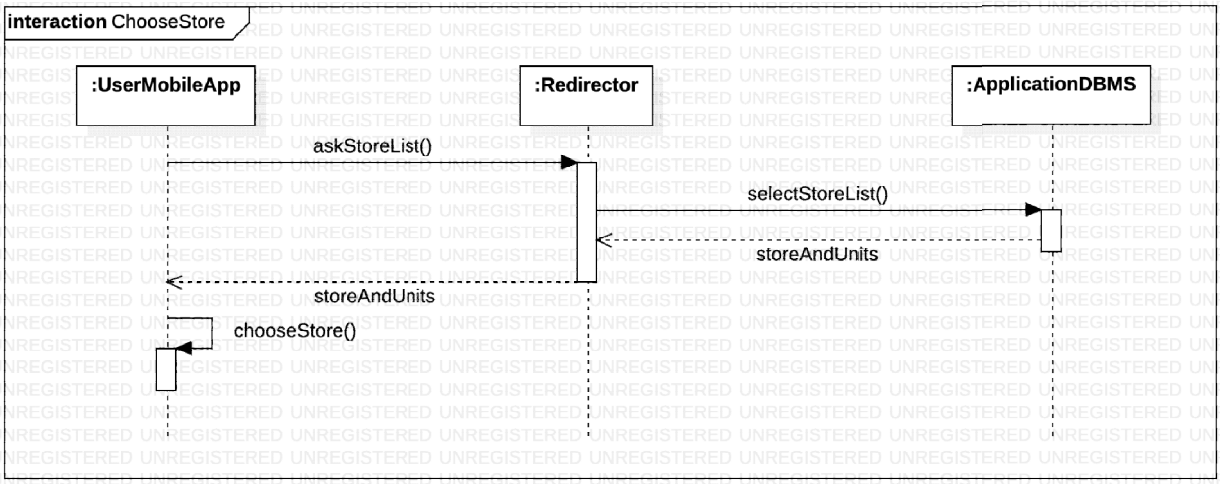


2.4.2 Choose store

Choose store is a section of the registration.
It has been divided from the rest of the runtime view for readability and reusability.
The choose store process allows the user to choose the store.

The mobileApp sends a request to the application. The request asks for the store list, it is handled by the Redirector which queries the DB through the applicationDBMS component.

The Application DBMS sends back the list of the supermarkets and the available units in each supermarket.



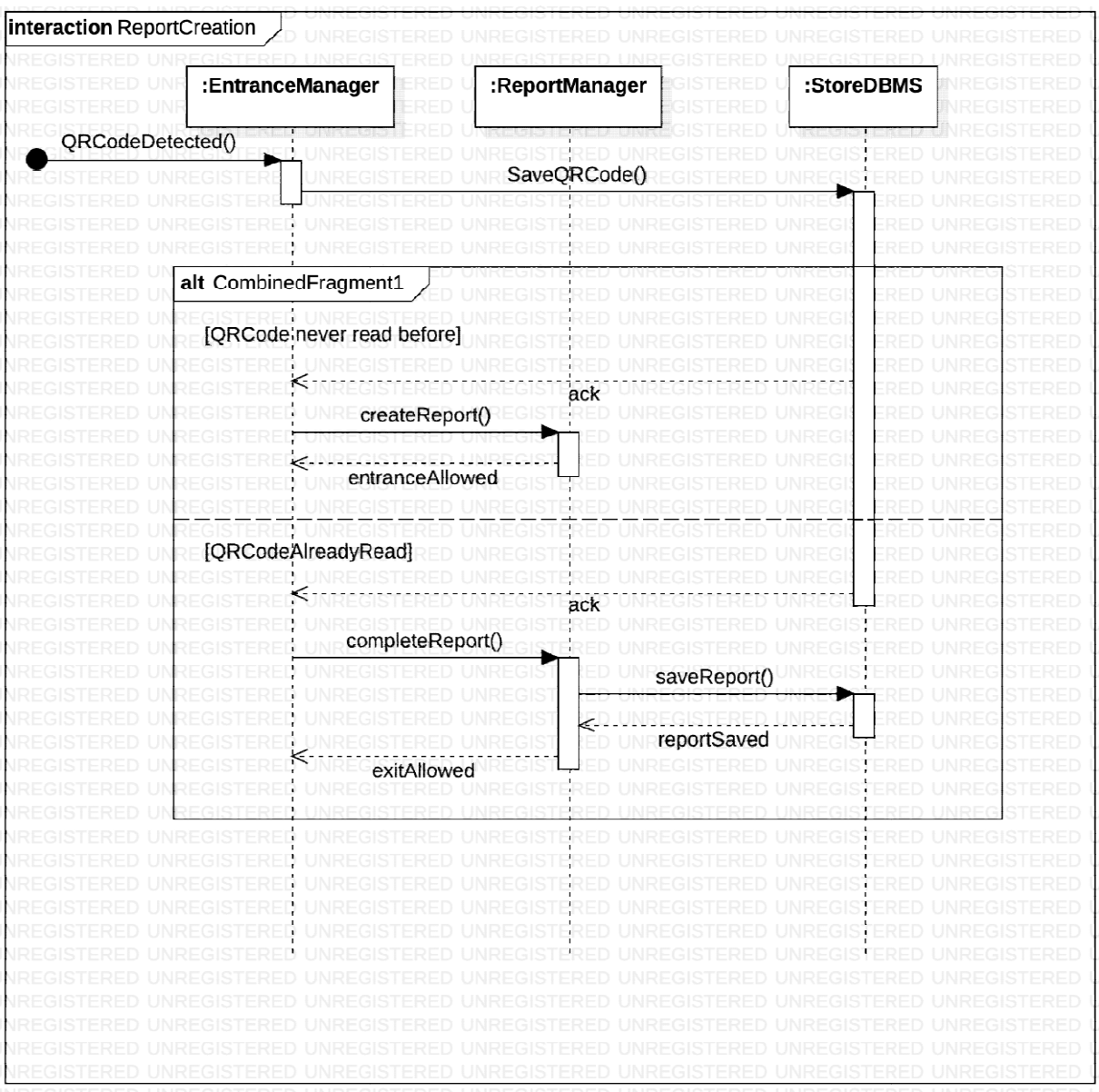
2.4.3 Report Creation

When a QRReader detects a QRCode checks if it has been previously read.

If no QRCode like the one that has been detected is saved in the DB then it is invalid and the person is not allowed to enter the building.

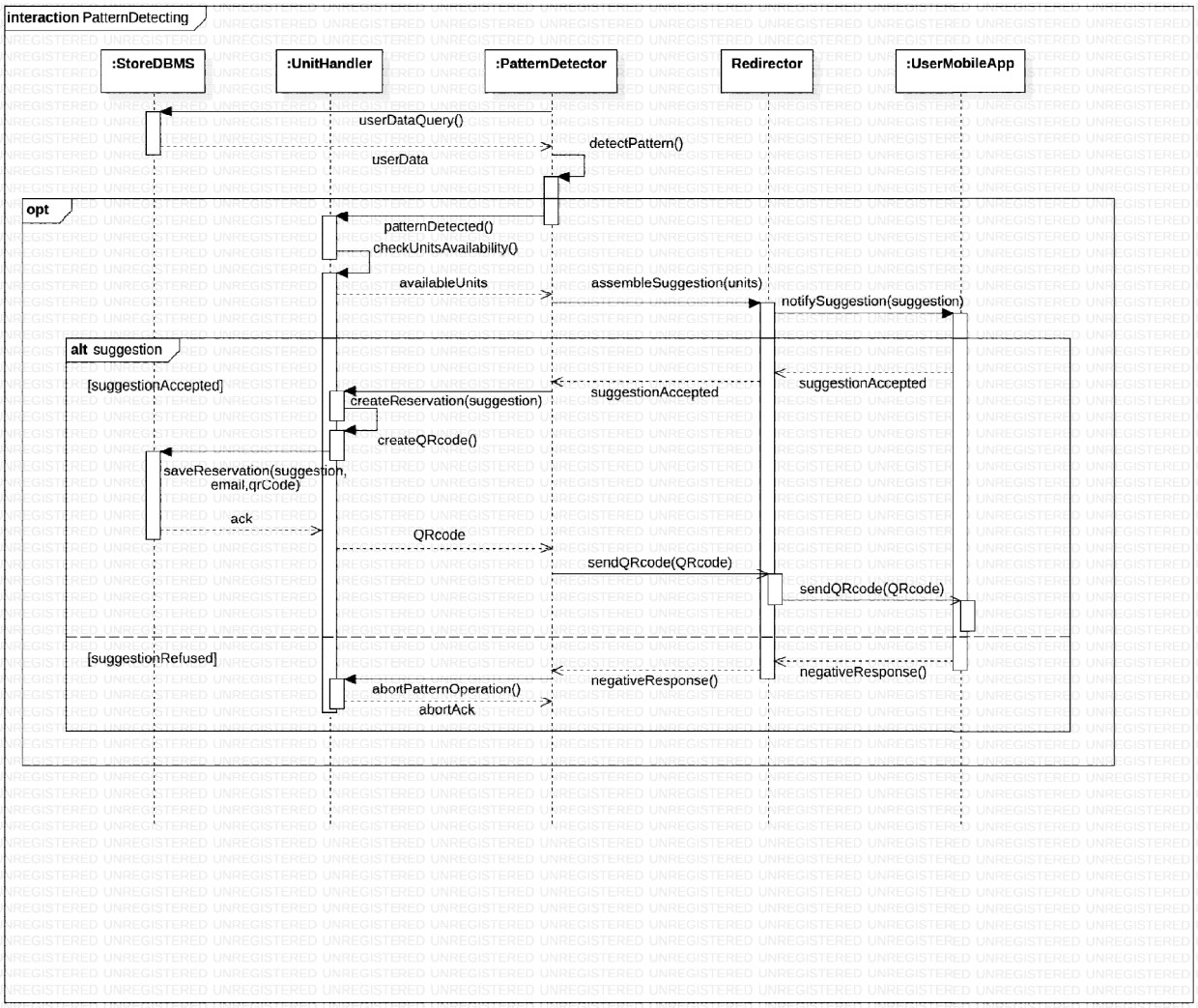
If the detected QRCode is valid but there is not a report then the person is allowed to enter the building and the report manager begins to create the report only if the person has reserved via the mobile app (Remember that the reservation made ad the utility desk are not reported).

If the detected QRCode is valid and the report has been initialized or the QRCode is associated to a reservation made at the store then the person is allowed to go out the supermarket and if it is necessary the report is completed.



2.4.4 Pattern Detector

The pattern detector asks for the users data. If it detects a pattern the unitHandler checks for availability of spots. In a positive case the pattern detector assembles the suggestion and send it to the user who can accept or refuse it.
In the case that the suggestion is accepted the unitHandler creates the QRCode send it to the user and saves it in the DB.

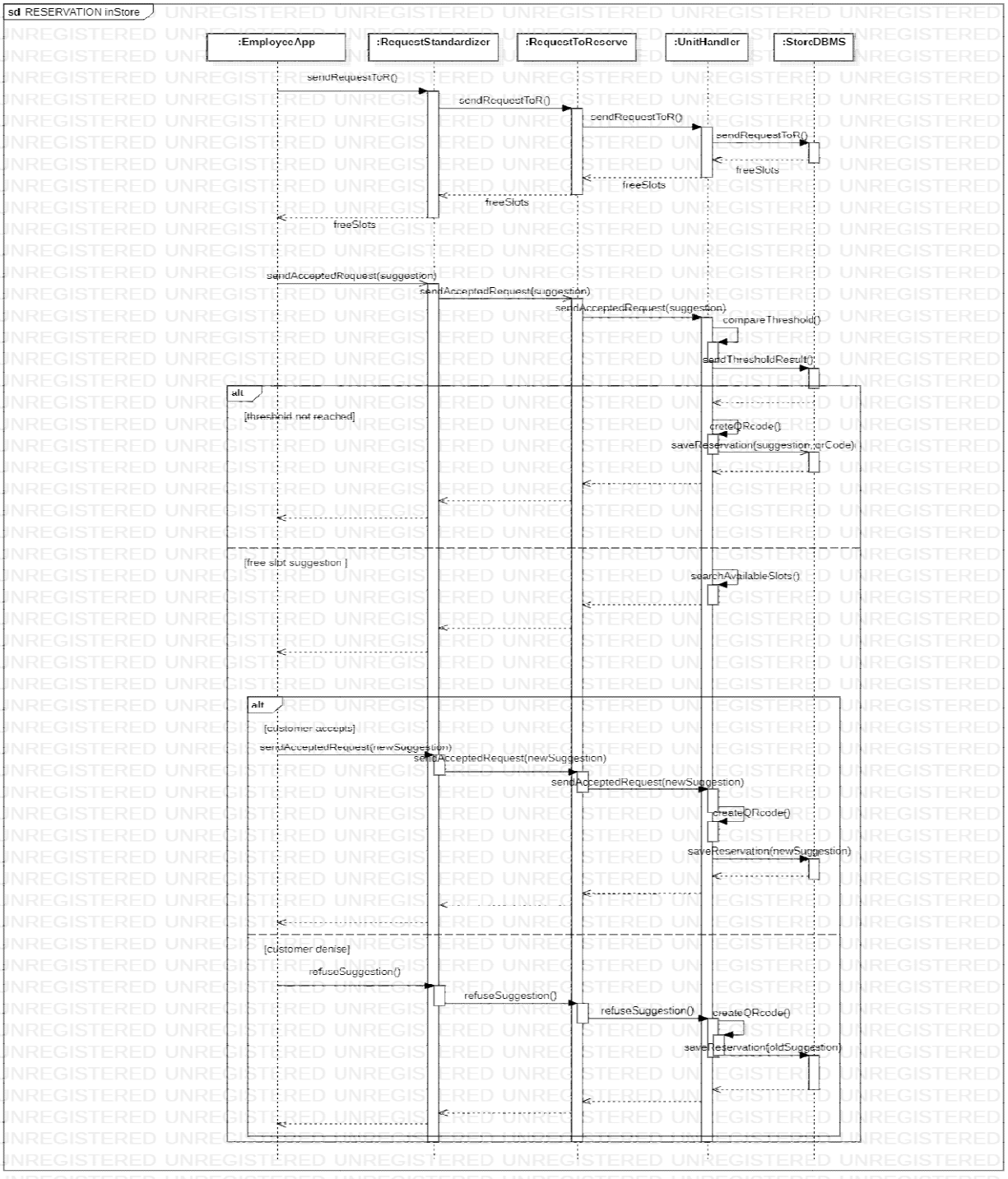


2.4.5 Reservation in Store

The employee app ask to reserve this request reaches the unitHandler which joint to the RequestToReserve compose the slotSuggestion which is sent to the user.
The user chooses the slot and sends it back to the reservationRequest

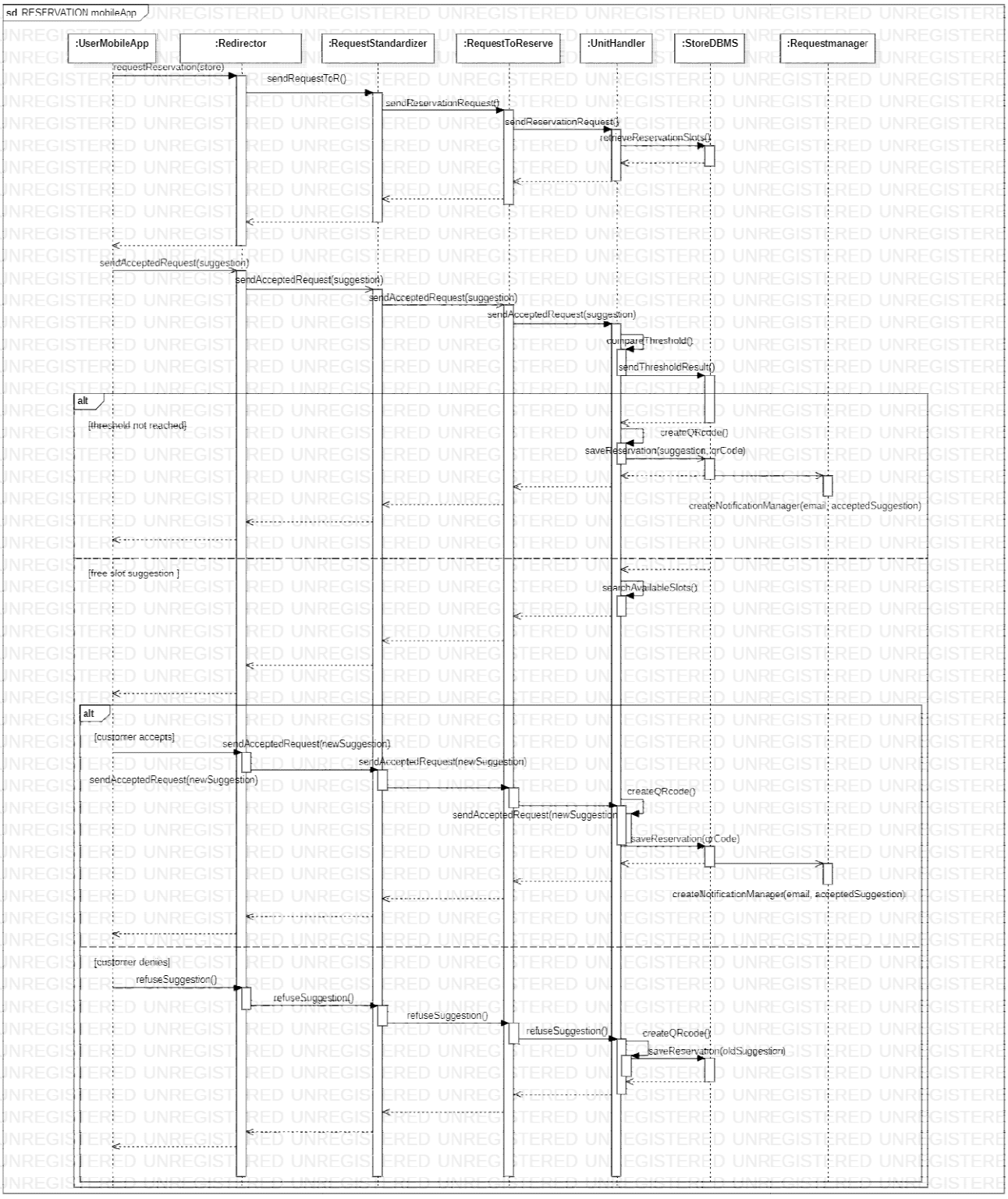
If the threshold is reached the system notifies the user which can change or confirm his/her reservation.

In any case a suggestion is choosen and confirmed or the operation is aborted.



2.4.6 Reservation MobileApp

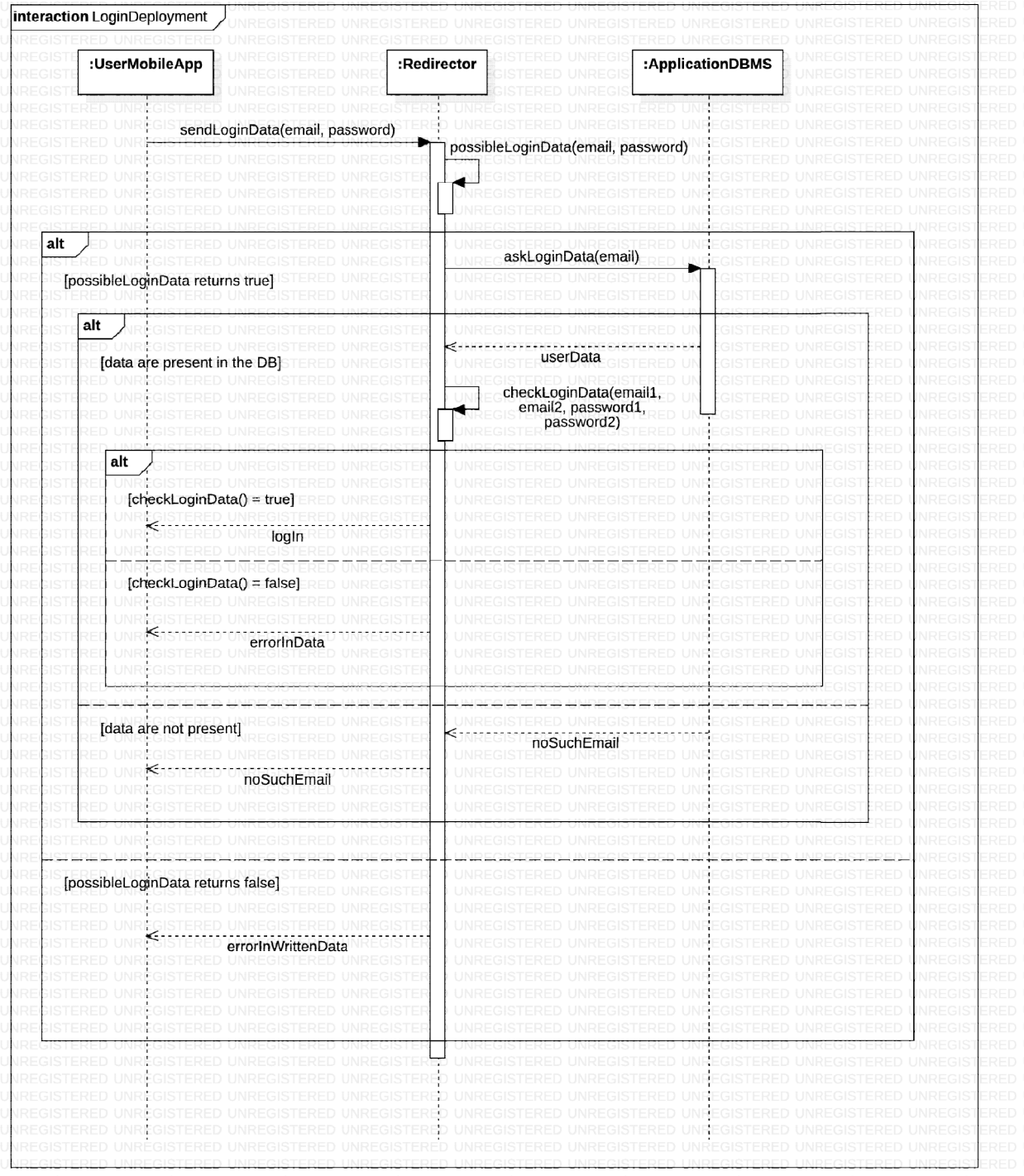
The interaction of reservation via mobileApp is the same of reservation in stor but the request is not sent by the employee ap but by the user mobile app.



2.4.7 Login Deployment

The mobileApp sends the login request alogn email and password.
The redirector checks if the sent data are feasible data and interrupts the procedure if for example the email or the password aren't sent.

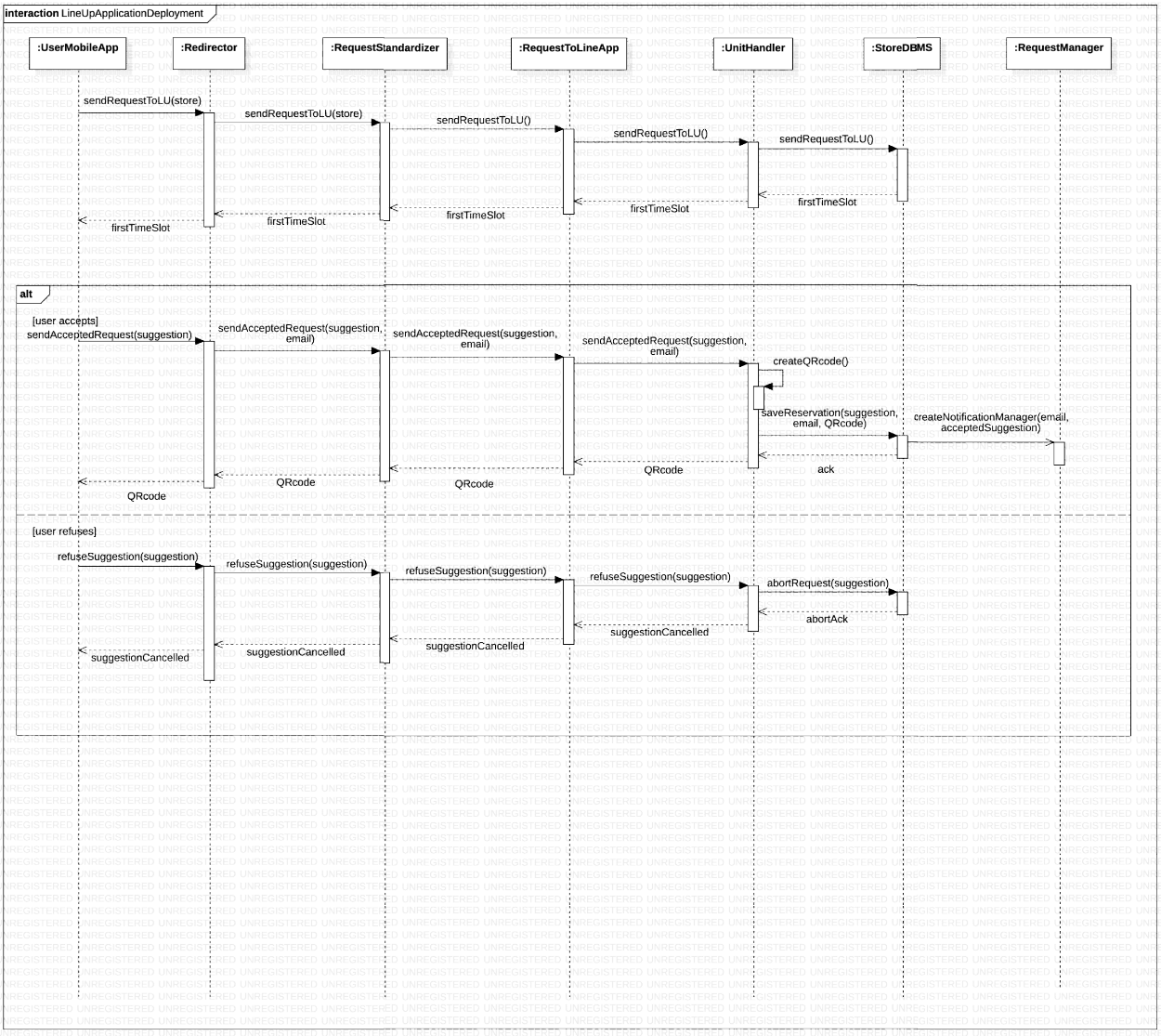
The redirector ask to the applicationDBMS the password saved in the DB corresponding to the email sent by the user. If the retrieved password is correct then the access is granted otherwise an error is generated.



2.4.8 LineUp MobileApp

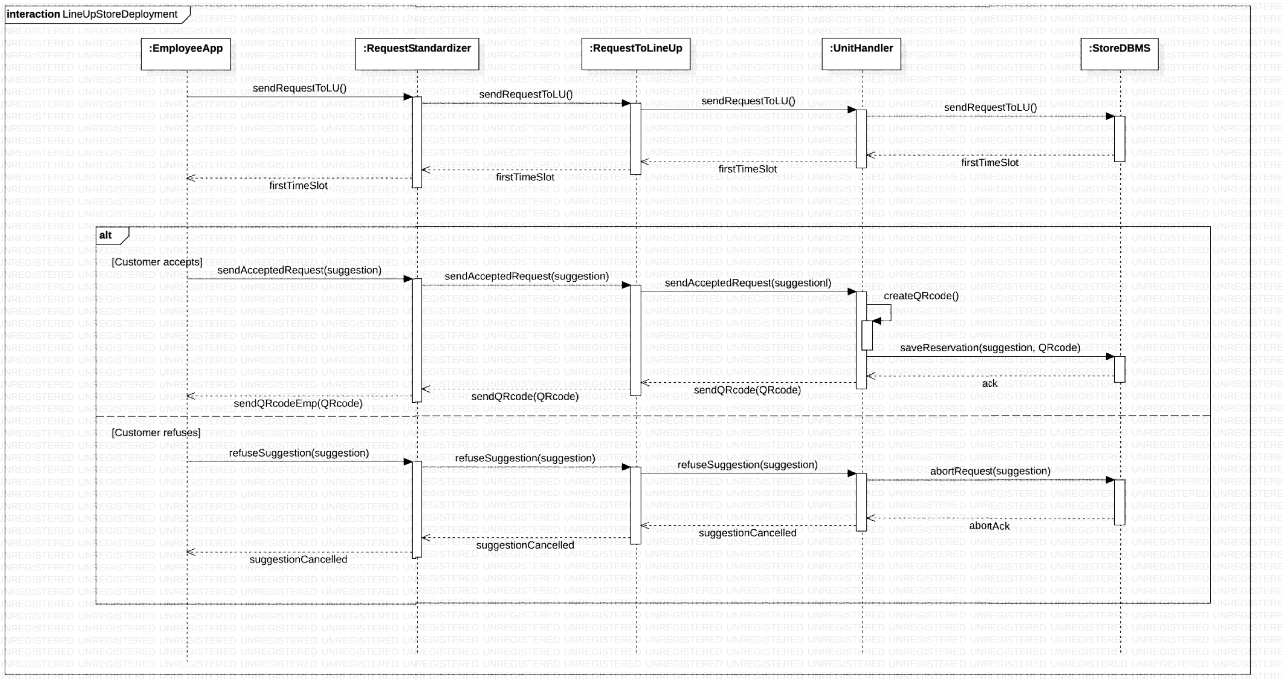
The userMobileApp send the lineup request to the redirector it submit to the requestStandardizer which checks the availability of a temporal slot thanks to the unit handler.

If a slot is fount it is sent to the user as suggestion he/she can accept it (and then it is saved as reservation) or refused (operation aborted)



2.4.9 LineUpStore

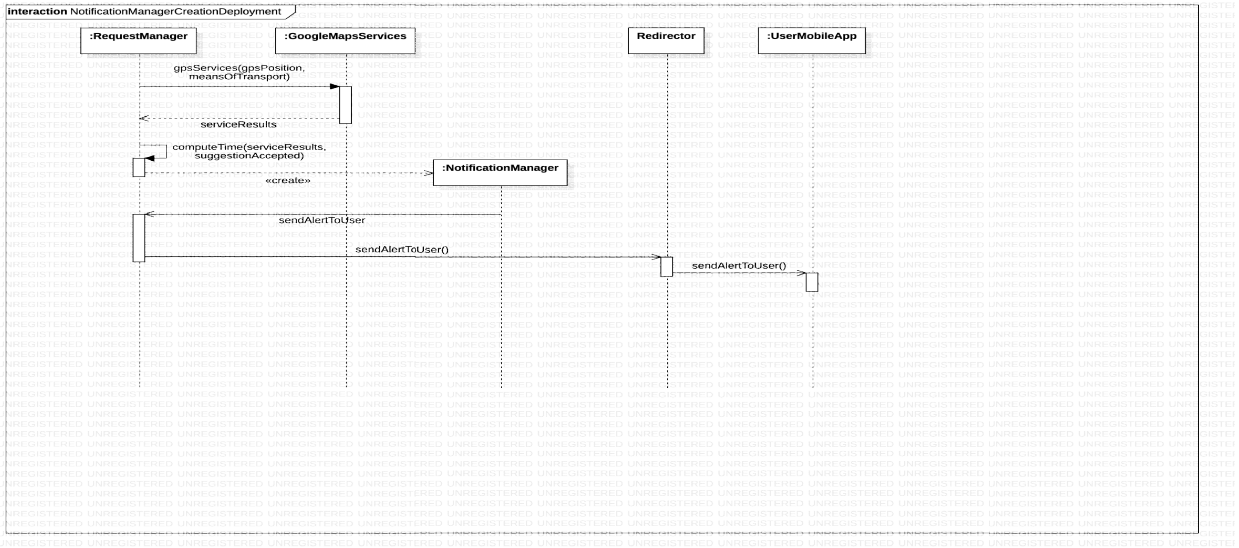
The system behaves in the same way as the LineUp mobileApp but the first component is the Employee app and not the user mobile app.



2.4.10 Notification Manager

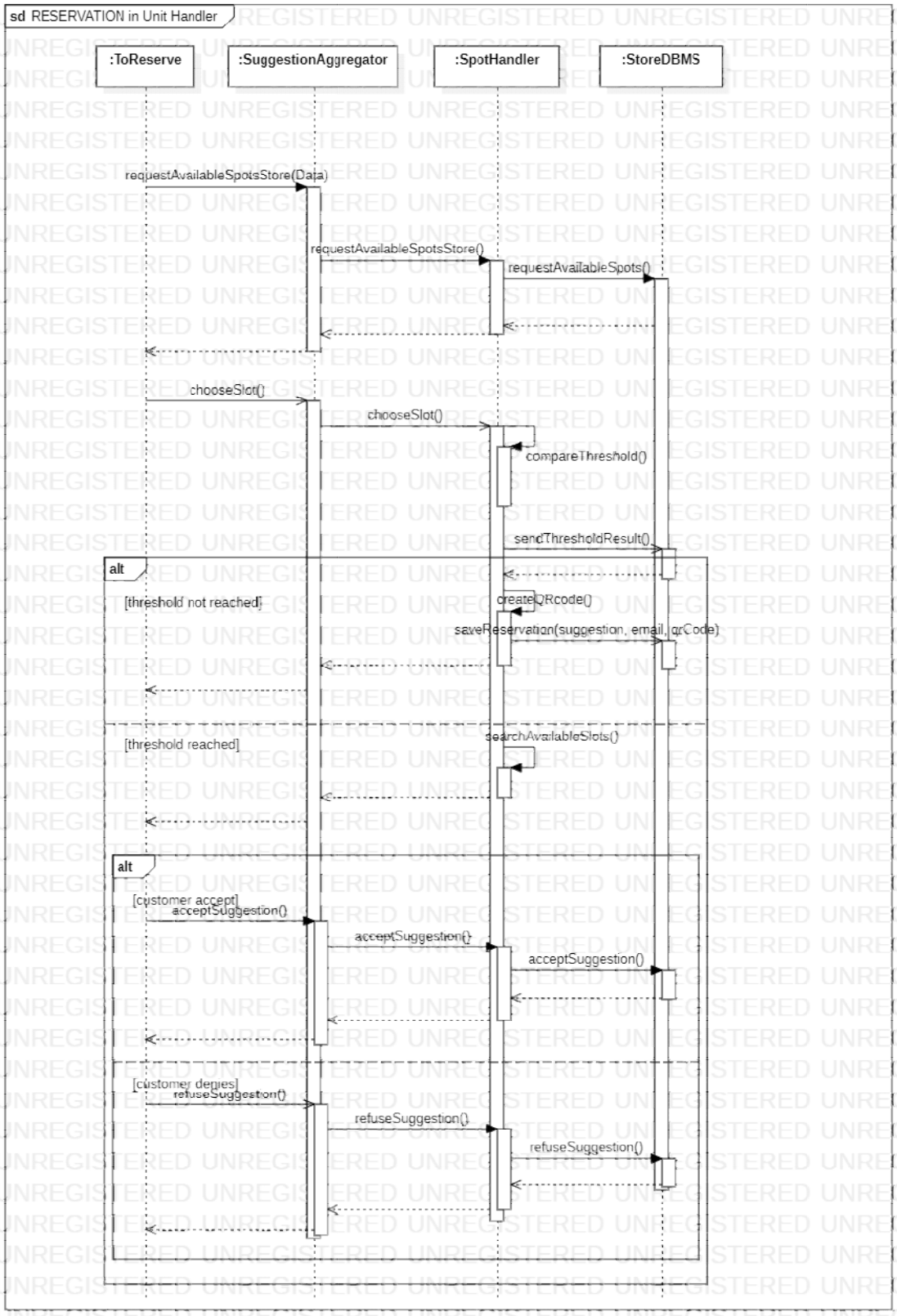
The RequestManager compute the required travel time of an user which has made a reservation thanks to the GoogleMapsServices component then a notification is created by the notification Manager.

The notification is sent to the UserMobileApp by the requestManager going through the redfirector.

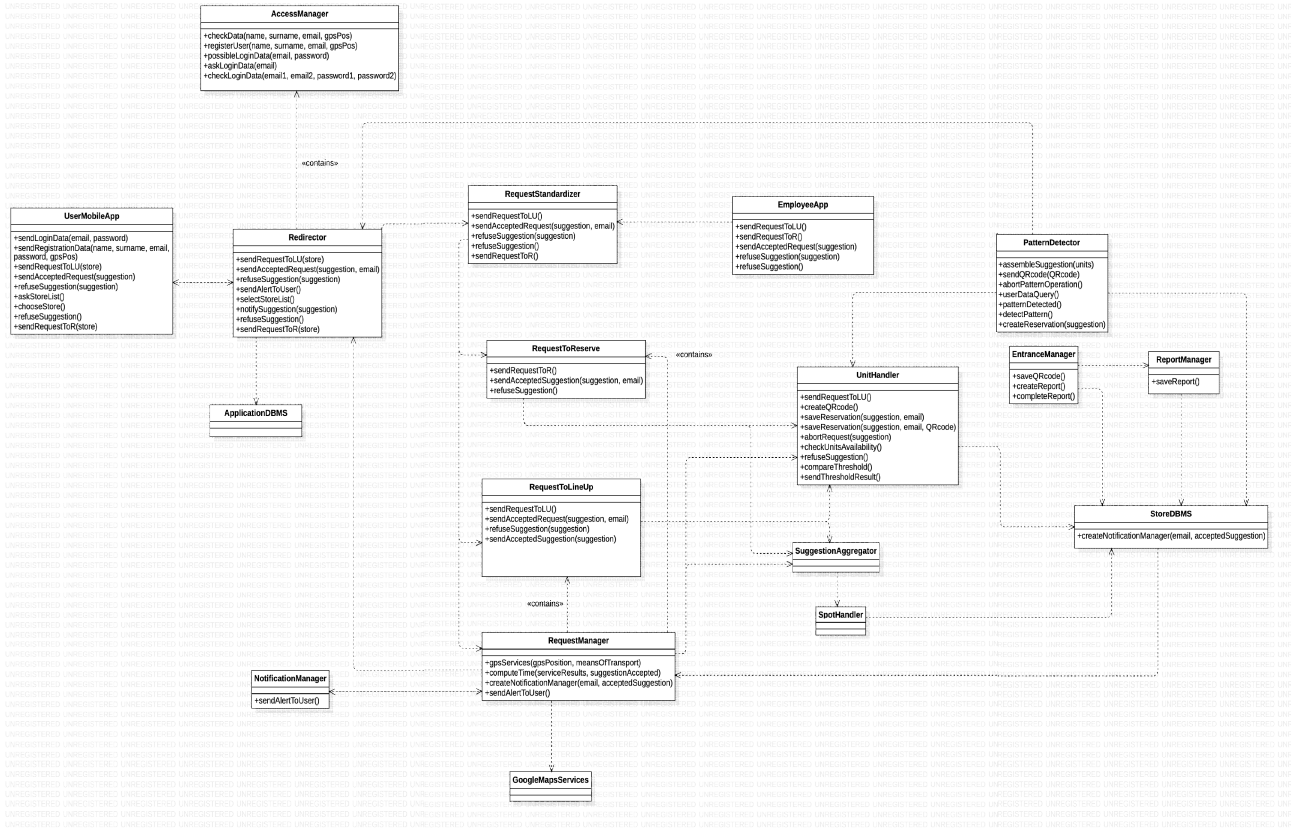


2.4.11 Reservation in UnitHandler

This interaction as choose store is a part of the reservation or lineup request.
It has been divided in order to keep the runtimeView more readable and simple.



2.5 Component interfaces



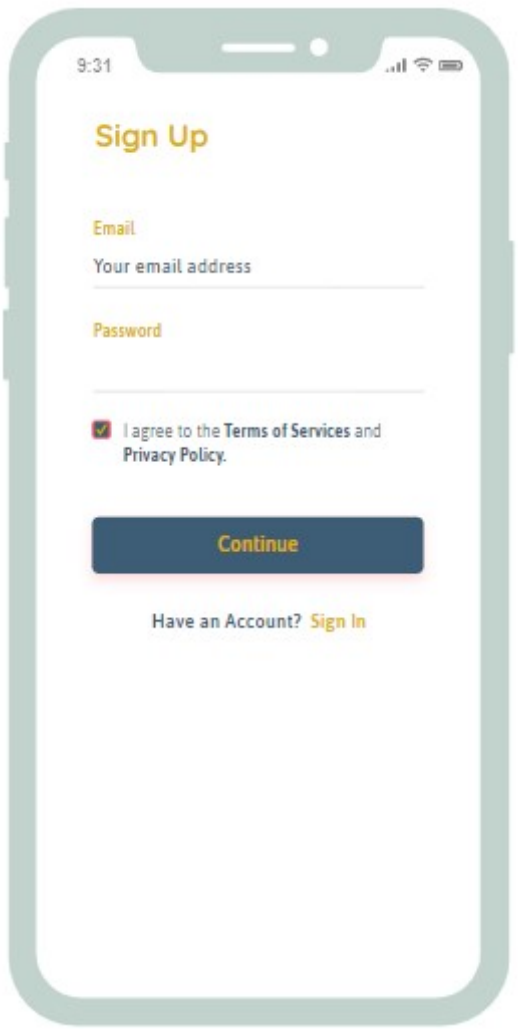
2.6 Selected Architectural Designs and patterns

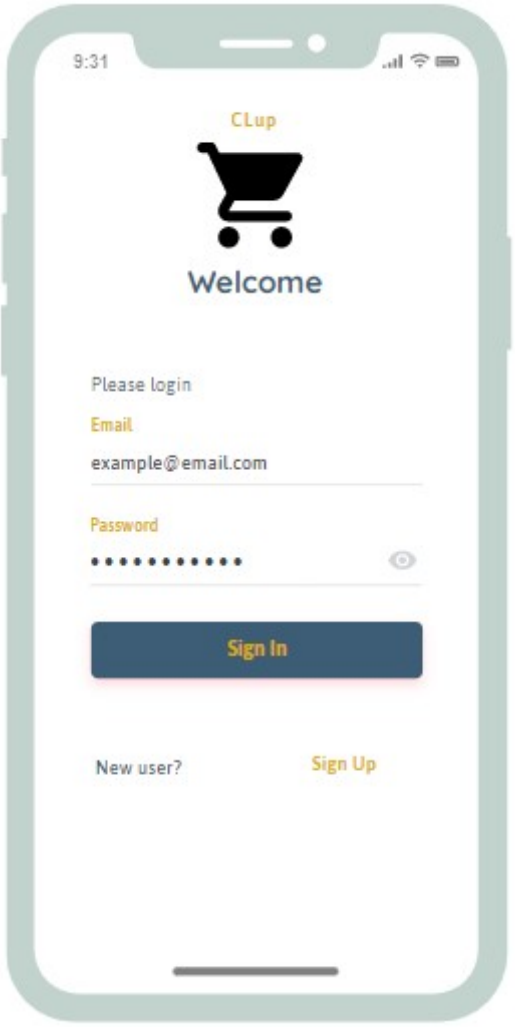
In this project have been used several design patterns the most relevant are:

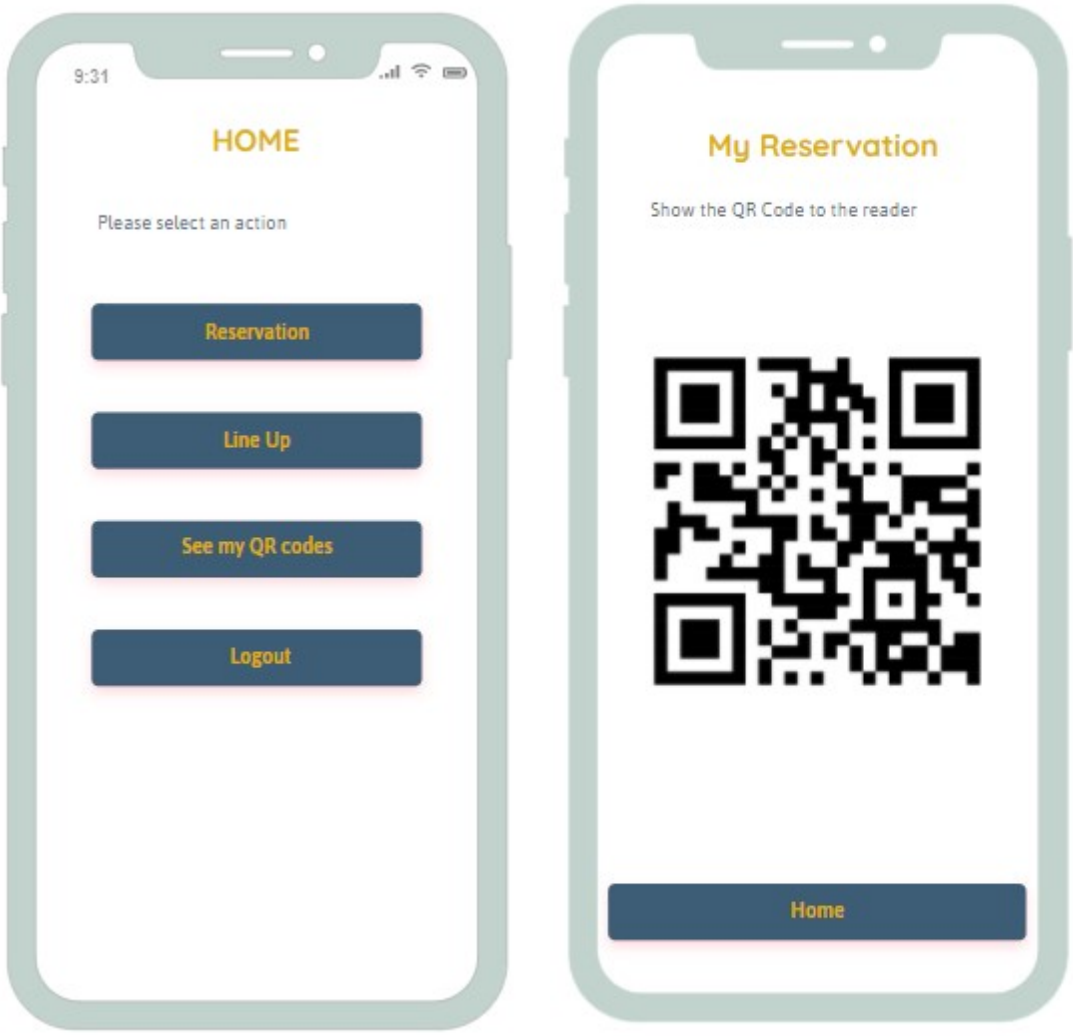
- MVC pattern where the model is entirely contained in the BD, the controller is in the business layer and the view is contained in the clients.
- Facade: used to implement the redirector component
- Publisher and subscriber: Used for the notification and pattern detection components (we have slightly modified the pattern in order to adapt it to the project).

3. User Interface Design

3.1 Mockups






9:31
HOME
Please select an action
Reservation
Line Up
See my QR codes
Logout
My Reservation
Show the QR Code to the reader
Home

4. Requirements Traceability

G1	Lining up mechanism is effective
	Requirements: R1, R2, R3, R4, R5, R6, R7, R9, R15, R16, R17, R18, R19
	• UserMobileApp • ApplicationDBMS • Redirector • EmployeeApp • Request Standardizer • RequestManager • NotificationManager • GoogleMapsServices • UnitHandler • SuggestionAggregator • SpotHandler • EntranceManager • ReportManager • StoreDBMS
G2	Customer can make requests by the utility desk of the store
	Requirements: R1, R2, R4, R5
	• EmployeeApp • RequestStandardizer
G3	Maximize the number of people in the store
	Requirements: R1, R2, R7, R8, R9, R10, R11, R12, R13, R14, R18, R19, R20
	• UnitHandler • PatternDetector • EntranceManager • ReportManager • StoreDBMS • NotificationManager
G4	Balance out the number of people in the stores
	Requirements: R1, R2, R5, R6, R7, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20
	• PatternDetector • NotificationManager • StoreDBMS • Entrance Manager • ReportManager • UnitHandler

R1	The system allows the user to request to line up, in order to go to the store.
R2	The system allows the user to book a visit at the store.
R3	The system doesn’t allow the user to line up or reserve a visit twice in a particular slot.
R4	The system allows the store employee to make multiple requests in particular slot.
R5	The system provides customers with a reasonably precise estimation of the waiting time.
R6	The system alerts customers taking into account the time they need to get to the shop from the place they currently are.
R7	The system can estimate the time of visit of a client.
R8	The system generates a QR code to allow people to enter and exit the store.
R9	The system fixes a standard time of visit for customers who approach for the first time.
R10	The system allows the user to insert the estimated time of visit.
R11	The system allows the user to insert the categories of goods he or she is going to purchase.
R12	The system fixes a standard list of categories as default.
R13	The system records the time that customers spend in the supermarket.
R14	The system saves the categories bought by the user.
R15	The system suggests alternative temporal slots.
R16	The system suggests alternative stores.
R17	The system notifies users of available temporal slots.
R18	The system allows the user to sign up to the application.
R19	The system allows the user to log in.
R20	The system doesn’t save any data of the requests made by the employee.

5. Implementation, Integration and Test Plan

5.1 Overview

In this part implementation, integration and test planning strategies are explained, in order to schedule an optimal verification process, improving the overall software quality.

5.2 Implementation

Implementation phase is carried out through multiple bottom-up processs.
In addition to implementation, also unit testing has to be carried out in an incremental way.

Since our system is structured into two main logical tier, one more general and one specific for a particular store, it has also two different data bases to deal with. It is convenient so to split the implementation into multiple disjoint processes:

- Two for the presentation layer: UserMobileApp and EmployeeApp. This is meant to let web designers focus on presentation only, leaving the work of integration to others.
- One for the first logic layer: starting from ApplicationDBMS to Redirector, in order to test properly flows of users' data.
- One for the second logic layer: starting from StoreDBMS to RequestStandardizer, in order to test the core business of the application, that is dealing with users' reservations and requests.

First two points of this list need no explanation of development order, since the sequence of components is restricted to one or two components and already described above.

The last one is the most critical part and necessary clarifications are written below:

- At first StoreDBMS has to be implemented, since every component deal with it (directly or indirectly). This component let the system handle data of reservations and users' requests on a single supermarket, interfacing the overall system to the store database.
- EntranceManager, UnitHandler, ReportManager and RequestManager can be then implemented, since they are the only components which communicates directly with StoreDBMS. The preferable order is the one written above.
- PatternDetector, NotificationManager and RequestStandardizer can be finally implemented: the former two represent functionalities which improve the quality and the overall experience for the user, the latter is indeed a real interface to communicate with the other logic layer.

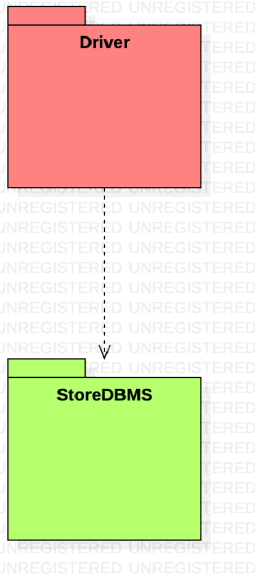
5.3 Integration

Due to the architecture of our system and the structure of the implementation strategy,

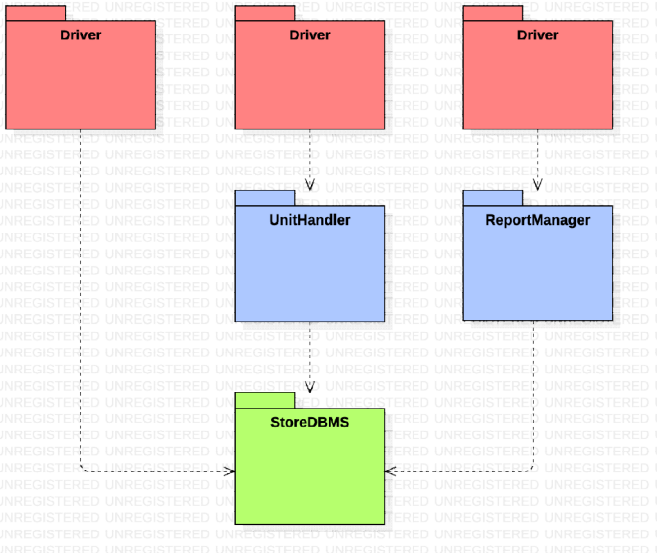
a bottom-up integration would be indeed easier to realize than other methods.
It has been however necessary to carry out a mixed strategy, in order to achieve a better integration among modules which need to be implemented by other teams (as seen for the first logic layer).

Here's a general schedule to proceed with the integration process:

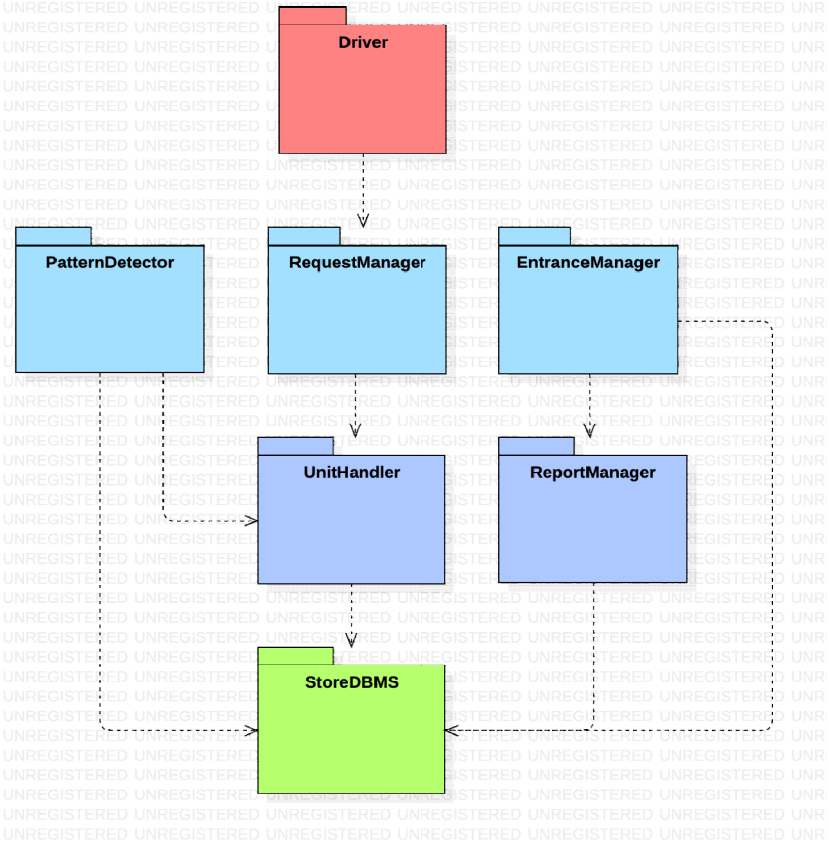
1. As for implementation, the starting point of the process is StoreDBMS. It is connected to a driver, as bottom-up strategy implies.



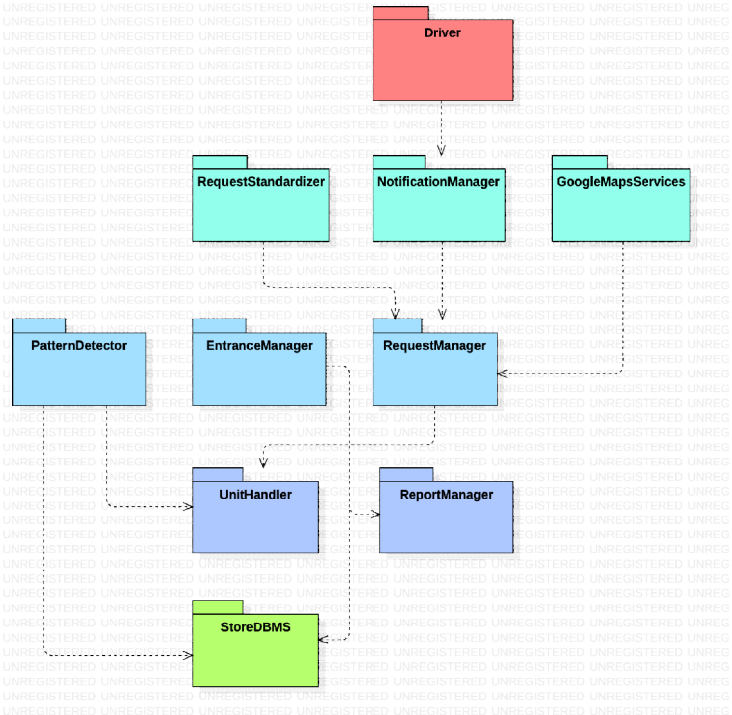
2. ReportManager and UnitHandler are considered then with their respective drivers. Note that StoreDBMS has a driver too, because of the fact EntranceManager depends on it and ReportManager and then PatternDetector too depends on it and UnitHandler. Their integration will be discussed later.



3. As told above PatternDetector and EntranceManager are integrated in this step, as well as RequestManager. Note that PatternDetector and RequestManager communicate with the first logic layer, so their integration with that part will happen at the end of all other integration processes.

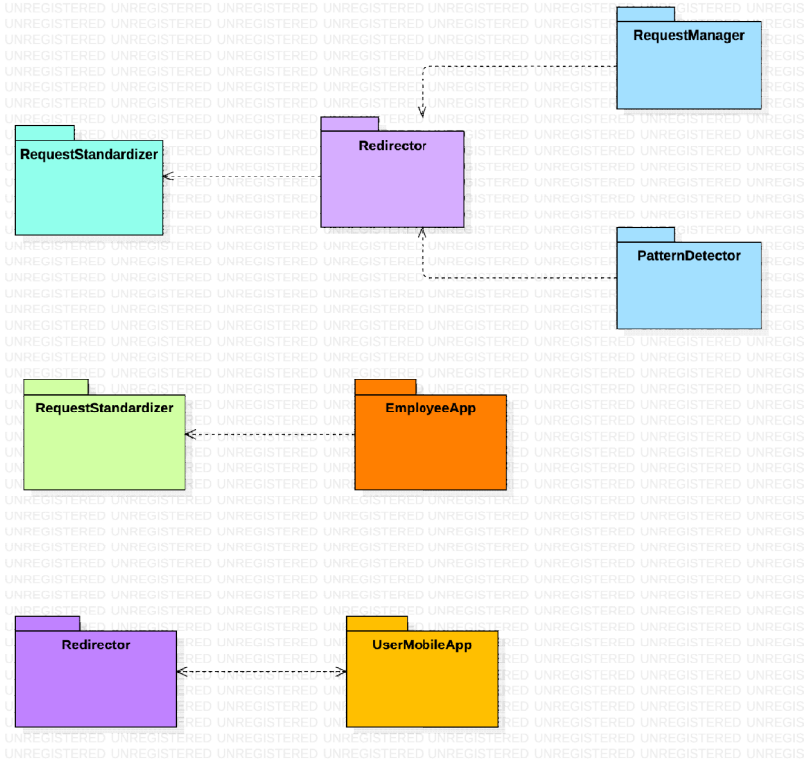


4. At this point, RequestManager is integrated with GoogleMapsServices, NotificationManager and RequestStandardizer. Finally, also NotificationManager is integrated well with its driver, that is RequestManager itself.



5. System's subparts are integrated:

a. At first, the first and the second logic layer are integrated. (assuming that the first layer too has been tested and integrated)
b. Then, logic layers are integrated with the presentation layer.



Note that all subcomponents are meant to be implemented and integrated as its container is too.

5.4 Testing

Once all components have been integrated, the next step to be done is the system testing.
This is a very critical part of the testing process, because of the presence of more system testing strategies, which handle with different capabilities of the system to be.

The first system testing which should take place is the functionality testing. This is because all other testing strategies lie on the real functions of the application.

Once ascertained that all functionalities of the application work properly, a performance testing should be provided, in order to possibly optimize algorithms and obtain a more efficient application.

The most critical part of the system testing process is the load testing.
This application is intended to work for a chain of stores, which could be great in the amount. In addition to this, the amount of people who go to a particular store is not uniformly distributed: peak-hours are a well-known problem of stores, and they could vary on a daily basis (let's think of Christmas holidays: the flow of people entering the supermarket could be severely huge at every hour).

An example of what said above: assuming that people begin to crowd stores around 2 weeks before Christmas and every hour the amount of people in the store is always the allowed upper bound N of people permitted, we should have a total number of requests on those days for that particular store equal to

$$14*N*hoursOfOpening$$

Considering that not all people would be able to reserve a place on those days, a load testing with a number of test requests greater than the amount written above with the formula should be considered. In other worlds, for testing the “Christmas time” requests we should consider that number as the lower bound of made requests.
Note that the number 14 represents the days equal to two weeks, which is the maximum temporal limit where the possible registrations are seen. Christmas time has been taken as an example because of the fact it is one of the most crowded time for supermarkets.

Another key-part of the system testing process is the stress testing.
Its structure should rely on what written above about load testing, adding to it lots of requests of cancelation, in order to see whether the system recovers after severe changes of data in a minimum amount of time or not.

6. Effort Spent

Work in group

20/12/2020	1h
29/12/2020	1h 17min +1h 29min
30/12/2020	2h
2/1/2021	2h 36min
4/1/2021	2h 22min
7/1/2021	2h

Matteo Panzeri

Chapters 1 and 2	4h
Runtime view	2h
Editing	4h

Riccardo Pellini

Sequence diagram	3h
Interface diagram	1h
Testing and integration	5h
Component diagram	1h

Paolini Giuseppe

Class diagram	2h
Sequence diagrams	4h 30 min
Mapping	1h

7. References

The diagrams have been made using StarUML.
The Mockups have been made using https://moqups.com/