POLITECNICO
MILANO 1863

Design Document

CLup – Customer Line Up

Rei Barjami – 10588572

Simone Bianchera – 10611808

Niccolò Borrelli – 10615638

Contents

1 – Introduction

1.1 – Purpose

The purpose of this document is to give more technical details than the RASD about CLup application.

This document is addressed to developers and aims to identify:

- The high-level architecture
- The design patterns
- The main components and their interfaces provided one for another
- The Runtime behaviour

1.2 – Scope

The project *CLup – Customer Line Up* regards a system able to manage automatically online reservation (booking or line-up) for the queue of a store. In particular, it includes:

- Registration and login function for all type of users.
- Scan service that allows the store's staff to monitor the entrances and the leavings.
- Reservations' creation function that permits customers and store staff to generate new and different requests with different parameters, such as the store selected, the time estimated and the departments in which the customer will go.
- Automatic management of the queue.
- Automatic notifies for the main events of the reservation, such as when to leave home and when to enter. The last one is integrated with a monitor component.
- Customer profile in which the customers can see the status of the reservation.

1.3 – Definitions, acronyms and abbreviations

1.3.1 – Definitions

- Line up Request: Request to be put in the queue of the grocery store without any booking
- Hand out Customer: Customer who doesn't have access to technology for the CLup

1.3.2 – Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- DBMS: DataBase Management System
- MVC: Model-View-Controller

1.3.3 – Abbreviations

- CLup: CLup – Customer Line up
- [Gn]: n-goal
- [Rn]: n-functional requirement

1.4 – Document convention

In component diagrams of section *2.2 Component View* the dependency relation between components is specified using two different graphical notations in order to make the schemes more readable: the first one directly connects the required interface with the provided one using a solid line, while the second one connects components via a dashed arrow. The two notations are equivalent.

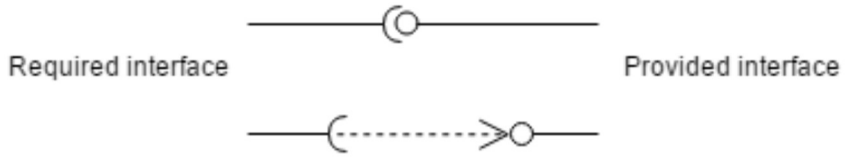


Figure 1- Dependency relation

1.5 – Revision History

Date	Version	Change
09/01/2021	1.0	

1.6 – References

- *IEEE 42010:2011 Standard for Systems and software engineering— Architecture description*
- *IEEE 1016:2009 Standard for Information Technology—Systems Design—Software Design Descriptions*
- *VerificationAndValidation on Beep.polimi.it*
- *https://www.uml-diagrams.org/*

1.7 - Overview

This document is composed by six part:

- **Chapter 1 - Introduction:** it explains the purpose and scope of the document, the conventions and the language necessary to understand the following parts.
- **Chapter 2 – Architectural Design:** in here different system views are provided, in particular it will focus on the component view, deployment view and runtime view.
- **Chapter 3 – User Interface Design:** this part will contain the mockups of the user interfaces.
- **Chapter 4 – Requirements Traceability:** it will provide a matrix containing the requirements and the components that realize them
- **Chapter 5 – Implementation, Integration and Testing Plan:** more details about the following steps of the project are shown such as: the order to implement components, integrate and test them.
- **Chapter 6 – Effort spent**

2 – Architectural Design

2.1 – Overview

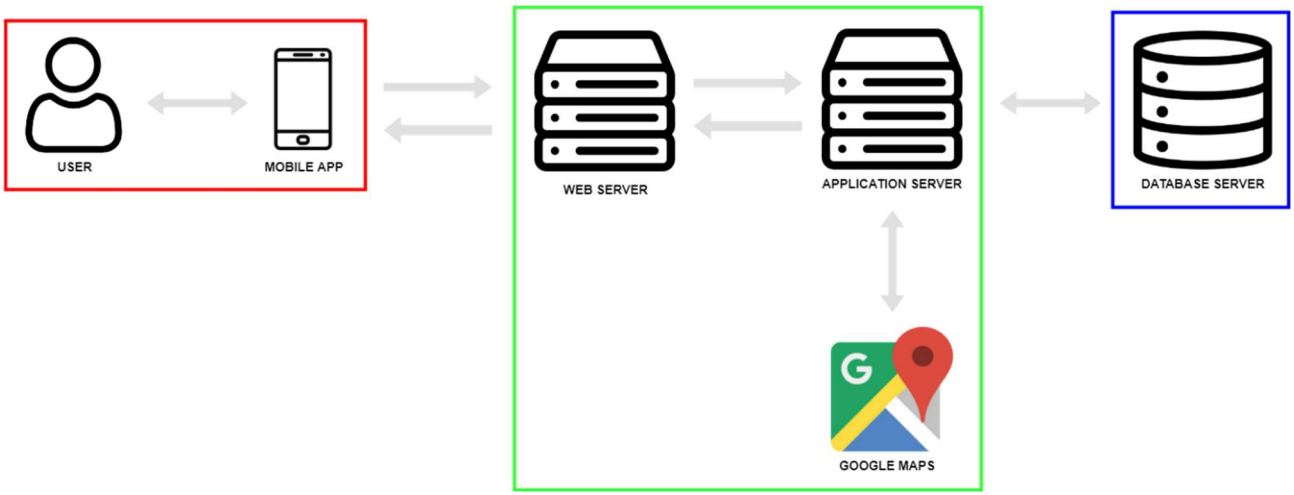


Figure 2 - Three-Tier Architecture

The architecture of the application is structured on three logic layers:

- **Presentation** layer handles the interaction with users
- **Application** layer controls the application's functionality, it makes logical decisions and moves data between the other two layers
- **Data access** layer takes care of the management of the information with the corresponding access to database

The application has to be made in client-server style due to the different physical machine between client and server. The communication among them takes place via other components and interfaces located in the middle of the structure, composed by hardware and software modules.

The application is a distributed application, therefore the layers described above will be installed in different hardware levels, called tiers. Modularizing the layer increases flexibility, scalability, development efficiency, reliability and availability as well.

2.2 – Component View

2.2.1 – Component diagram

In this section, it will be illustrated the component view of the application, how it's organized and all the parts will be briefly described in order to illustrate all the functions of the architecture.

Here below, the general component diagram is showed.

To make the diagram clearer, a set of colours is used to define which components and subsystems are part of the application server, which is in the DBMS server and which is external.

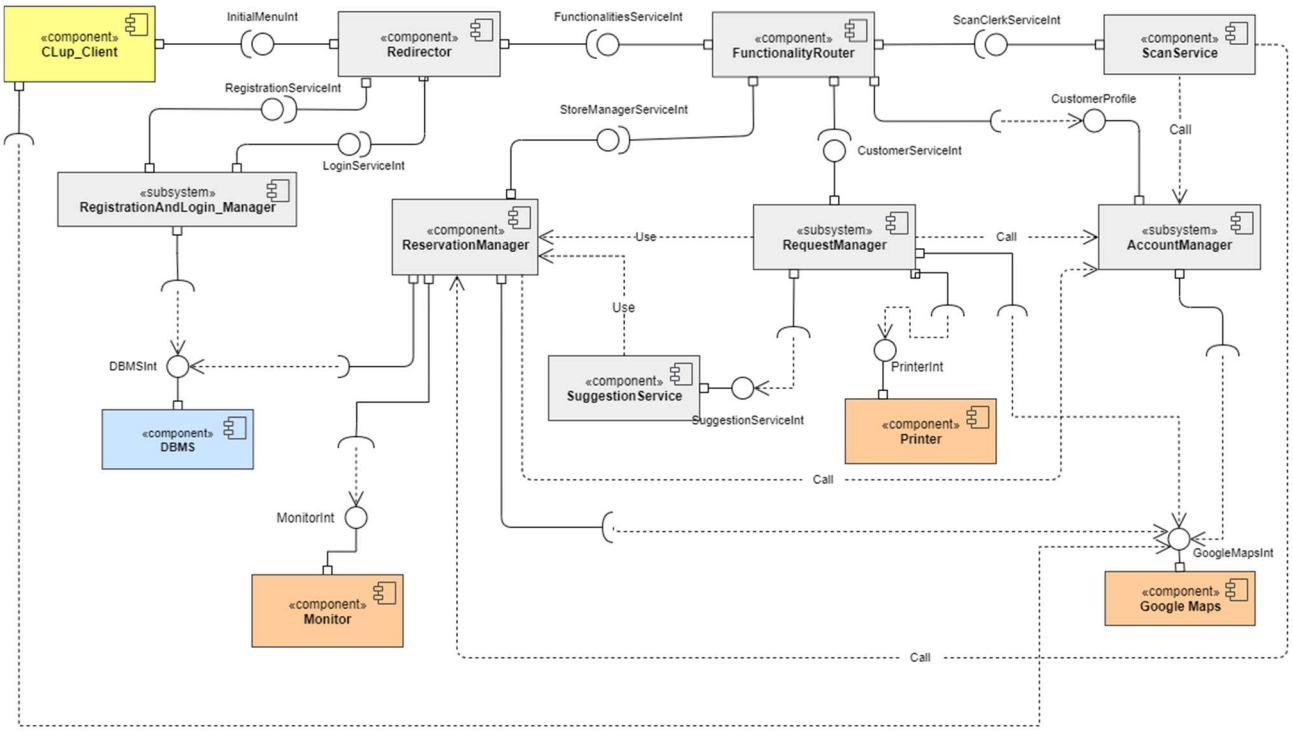


Figure 3 - General component diagram

The CLup_Client (in yellow) represents the mobile application. It isn't obviously a part of the application server. It's the application downloaded by the all the people that will use it, also for different purposes. In substance, the store managers, the clerks (hand-out and scan) and the customer will download the same application. The difference in the functionalities will be provided by the application server after the authentication of the client.

The first component that the mobile app will connect to is the *Redirector*. Its main function is to provide the initial menu for the clients. In case of unregistered or non-logged user, the *Redirector* will redirect the user to the correct interface, such as *RegistrationServiceInt* or *LoginServiceInt*. Otherwise, it will access to the *FunctionalityRouter* that will be explained in a second moment.

In the first scenario, the Redirector will connect the user (depending on his/her choice) to the *RegistrationAndLogin_Manager*.

The user will be connected to *Google Maps* to communicate his/her position during the request.

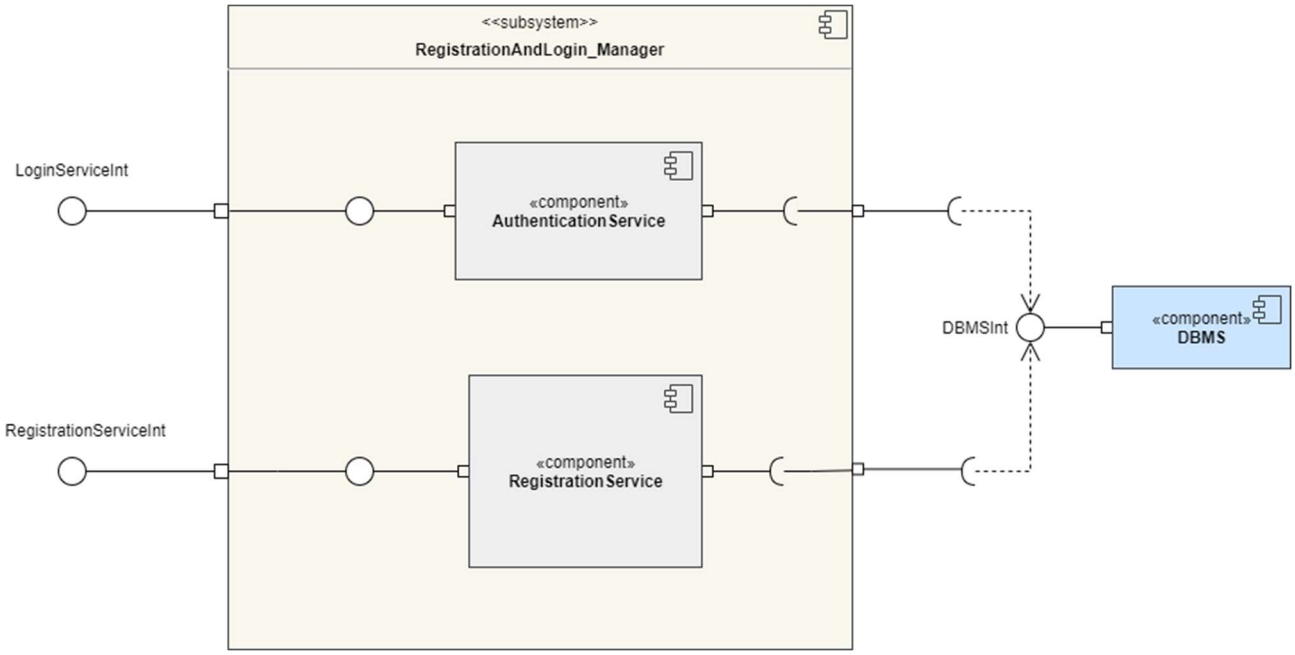


Figure 4 - RegAndLogin_Manager

This subsystem is necessary to guarantee a correct registration and authentication of the users. The control is made on the DBMS using a deleting connector to that because the behaviour is realized with it. The interface on which the subsystem will connect is the same for both services: in one case the existence of a tuple will be controlled, in another one a tuple will be added (if correct).

Once the *FunctionalityRouter* is reached a set of functionalities are provided, depending on the type of user that are logged.

The *FunctionalityRouter* allows:

- The store managers to access to the *ReservationManager*: she/he can see the data extrapolated from the queue, such as the number of customers that did line-up request or booking request and how many of them are inside the store.
- The customers to access to the *RequestManager*: he/she can formulate new requests or cancel his/her requests done before. The customer can also visit his/her own account profile through *AccountManager*, in which all the reservations done are shown.
- The hand-out clerk profile to have the same privilege of the customer with some limitations removed.
- The scan clerk profile to access the *ScanService*, a component that permits the scanning of the code in the entrances and in the leavings.

ReservationManager is the component which the main function is to organize the queue/list, to decide which is the next customer that will enter the store. It's done also knowing the time in which the customer will reach the store and it's calculated using *GoogleMaps* by the *RequestManager* during the creation of the reservation. The availability of the store for a created reservation that has to be evaluated is done by the *ReservationManager*. Moreover, the *ReservationManager* will save every reservation concluded in the *DBMS*.

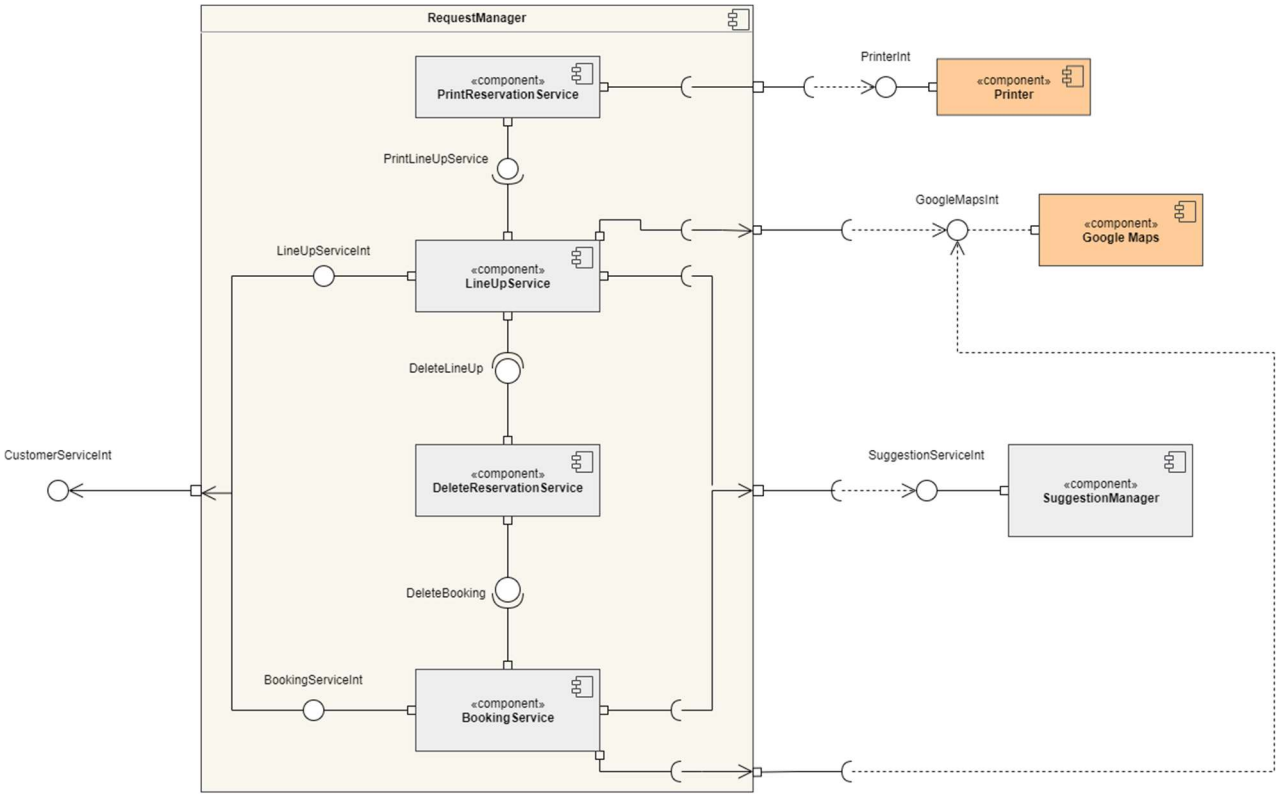


Figure 5 - RequestManager

The *RequestManager* offers all the services about making a new request or deleting an existing one. For the first version of the system only the hand-out clerk could print the reservations. This subsystem is also responsible for accepting or rejecting the request done due to availability provided by *ReservationManager*. If a request is rejected, the *LineUpService* and *BookingService* suggests the customer with other possibilities. This part is delegated to the *SuggestionManager*.

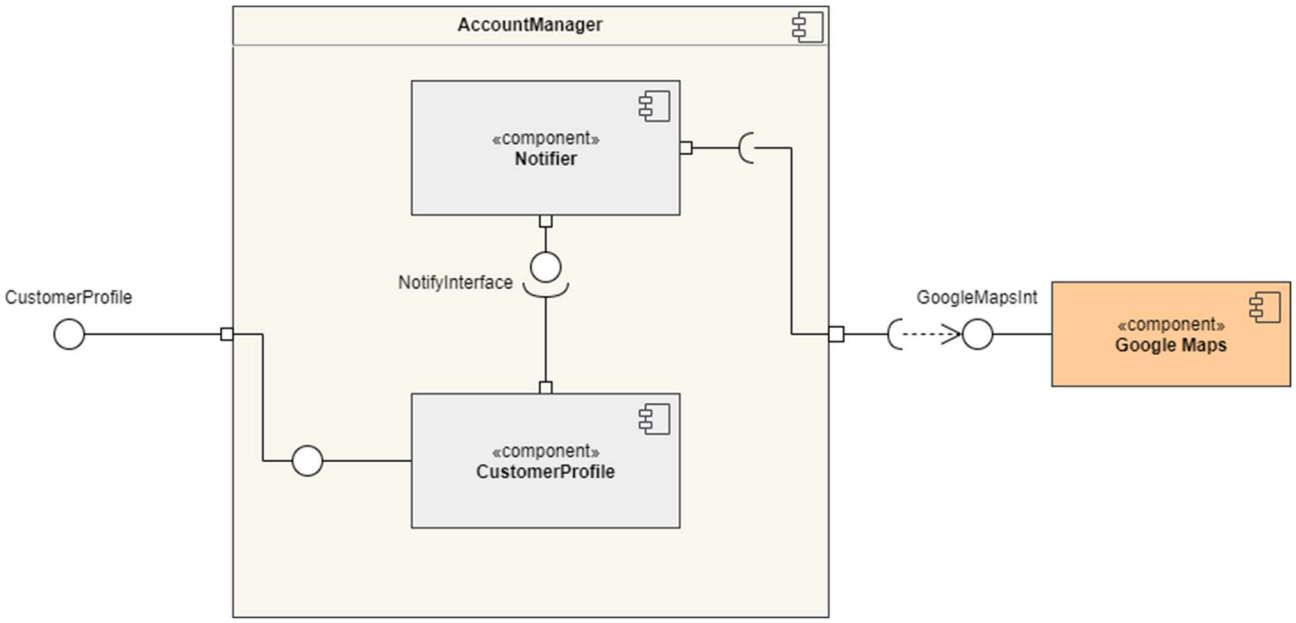


Figure 6 - AccountManager

Finally, the *AccountManager*, as written before, contains all the information about the customer in the *CustomerProfile*. In addition, it's also responsible for the notifies that have to be sent to the customer, such as: the acceptance of a request, the moment in which go to the store and when to enter. It has the access to *Google Maps* to be in touch with the change of the position of the customer (obviously not simultaneously).

2.2.2 – Component Interfaces

Below an interface diagram will show the dependency between them and an explanation will be provided.

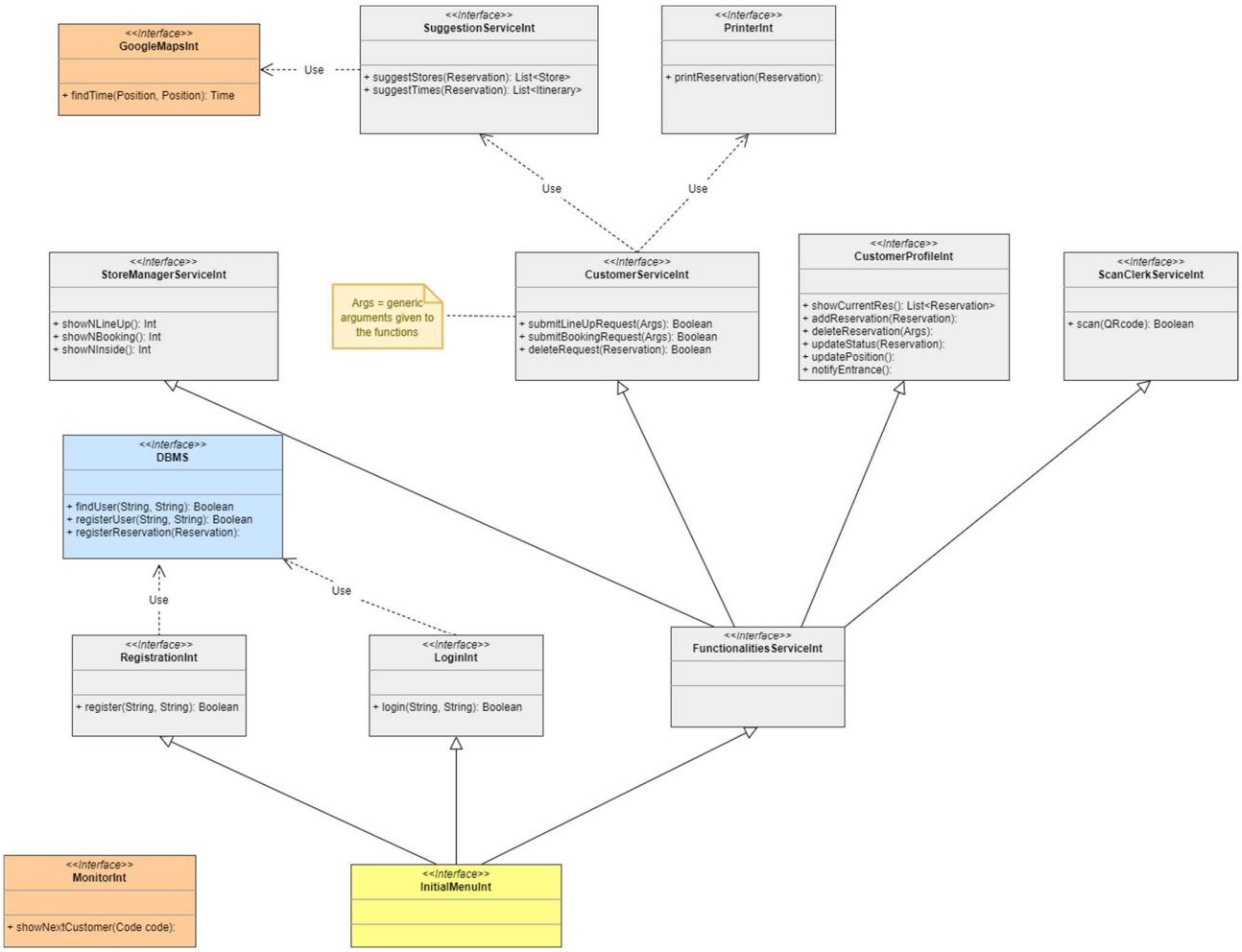


Figure 7 - Component interface diagram

Firstly, the *InitialMenuInt* is the only one with a direct contact with the mobile app (*CLup_Client)*, so he/she can access only to it. For this reason, this must permit the user to access all the possible functionalities the system offers. It's the subclass of other three main interfaces: one dedicated to the login (*LoginServiceInt)*, one to the registration (*RegistrationServiceInt*) and the last one to the effective functionalities. This choice explains the same structure of *FunctionalitiesServiceInt*: it's the subclass of all the interface that can provide all the services: *StoreManagerServiceInt, ScanClerkServiceInt, CustomerProfileInt* and *CustomerServiceInt.*

The last one written before can use the *PrinterInt* only if the user is the HandoutClerk. In further versions this structure allows the developers to make the printing possibility able also for the customer. Only the customer can use *SuggestionServiceInt* because the for the HandoutClerk a possible suggestion is useless.

The *SuggestionServiceInt* will use *GoogleMapsInt* in order to create convenient suggestions for the customer.

The *MonitorInt* isn't linked with other interface because no one from mobile application can access to that. It will be accessed internally by the server to show which customer will be the next to enter.

2.2.3 – Entity – Relationship diagram

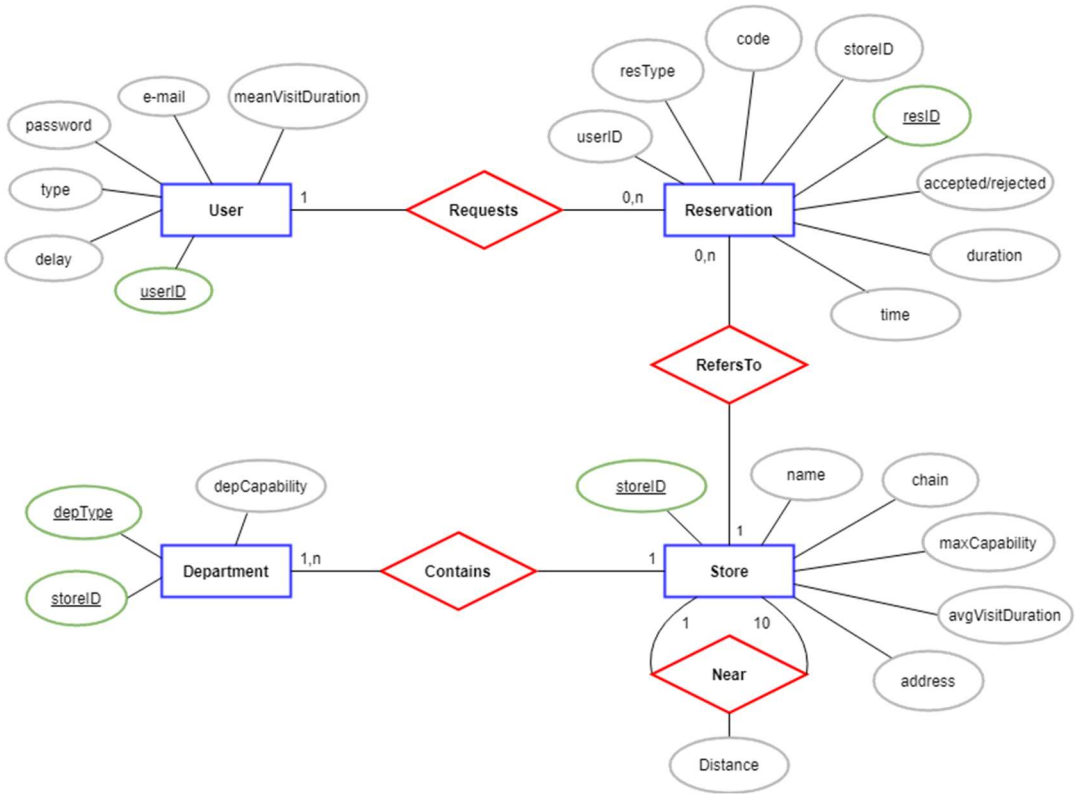


Figure 8 - Entity-Relationship diagram

2.2.4 – Additional specification about component view

A table represented the taken choices on the design will be shown and explained.

Name	Type	Purpose
User	JPA Entity	It will represent a user after the login
Store	JPA Entity	It will represent a store contained in the database
Department	JPA Entity	It will represent a department contained in a store
Reservation	JPA Entity	It will represent a reservation that can be store in database (usually detached)
Redirector & FunctionalityRouter	Message Driven	These components will redirect the user to the correct component depending on his/her type
RegistrationAndLogin_Manager	Stateless	To manage registration and login by user
ScanService	Stateless	To manage entrances by the scanning of the code
RequestManager	Stateless	To manage the creation of the request and the availability of that
SuggestionService	Stateless	To manage suggestion if a request is rejected
ReservationManager	Stateful	To manage the queue and the customers turn
AccountManager	Stateful	To manage the change the information about the customer

Redirector and *FunctionalityRouter* are message driven because they will receive messages from the queue and they will be activated in order to provide the correct functionalities to the requests.

RegistrationAndLogin_Manager, *ScanService, RequestManager* and *SuggestionService* are all stateless because they won't conserve any information, in particular the first one will check if the user is registered or register him/hers without storing any of that, passing all the data to the *DBMS*. The second and the fourth one are services provided by the server that allow different functionalities, such as the entrances, the leavings and the suggestions. None of them will store any data. Finally, the *RequestManager* is only responsible to create the requests, *AccountManager* and *ReservationManager* will manage them. Moreover, *ReservationManager* will contain all info about the store.

2.3 – Deployment View

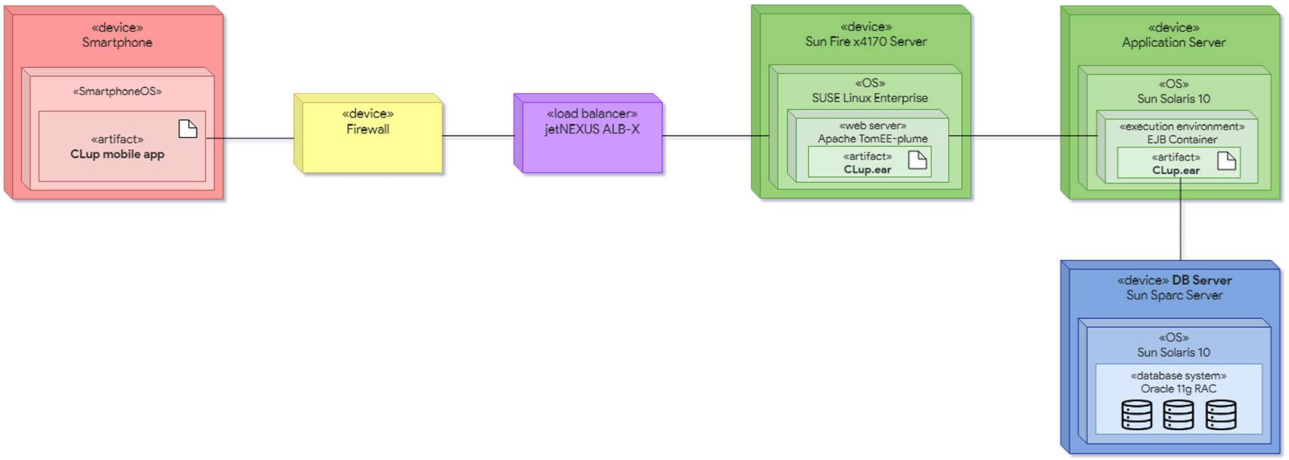


Figure 9 - Deployment Diagram

In the deployment diagram in the figure above the most important components are shown. Furthermore:

- **Smartphone**: customers, store managers and scan clerk will all use the app on the smartphone to perform any action that will be send to the application server through HTTPS
- **Firewall**: provides safety access to the internal network of the system as part of the safety of the system against external attacks
- **Load Balancer**: Distributes the workloads across multiple servers to increase capacity (concurrent users) and reliability of applications trying to avoid the overload of any single server.
- **Web Server**: Receives contents and requests from clients through the app and sends data received, to the Application Server for processing.
- **Application Server:** Here we have the application logic, not all because also the client has a part of this. The Application Server handles all the requests and provides the appropriate answers for all the offered services. It is directly addressed by the mobile application and it is replicated to avoid a single point of failure and to guarantee a better performance
- **DB Server:** Oracle Real Application Clusters (RAC) has been selected to perform the job of the DBMS. A cluster comprises multiple interconnected computers or servers that appear as if they are one server to the end users and applications. Oracle RAC enables you to cluster Oracle databases and it is a configuration of Relation DBMS (RDBMS). There are some nodes that contain the same instance of the database and the instances are connected to each other and share the cache in the Global Cache (GC). Then, there is a pool of replicated databases that contain datafiles. Datafiles are common, shared and they are accessed from all instances in a parallel and synchrony way. The choice of using Oracle RAC is due to provide performance, scalability, resilience and high availability of data at instance level. Performance comes from the load balancing because multiple machines are holding Oracle software, so it creates a better performance and the load is shared between all these servers. Scalability is due to the fact that it is possible to add more machines to hold Oracle software thus you are adding more nodes to the existing cluster. Resilience is provided: if one of the machines fails, that one could be taken down without bringing down the whole environment. Then the application will still be running thanks to the communication with other replicas. So, one machine could be brought down and fixed while others are running, hence this also brings to another important point that is the maintenance. In addition, to provide a higher availability we combine the RAC architecture with that of Data Guard that offers the functionality of Disaster Recovery. Data Guard has a working instance (called Primary) and one or more sleeping instances (called Standby) that are continuously updated and maintained aligned to the Primary instance in order to take his place in case of fail.

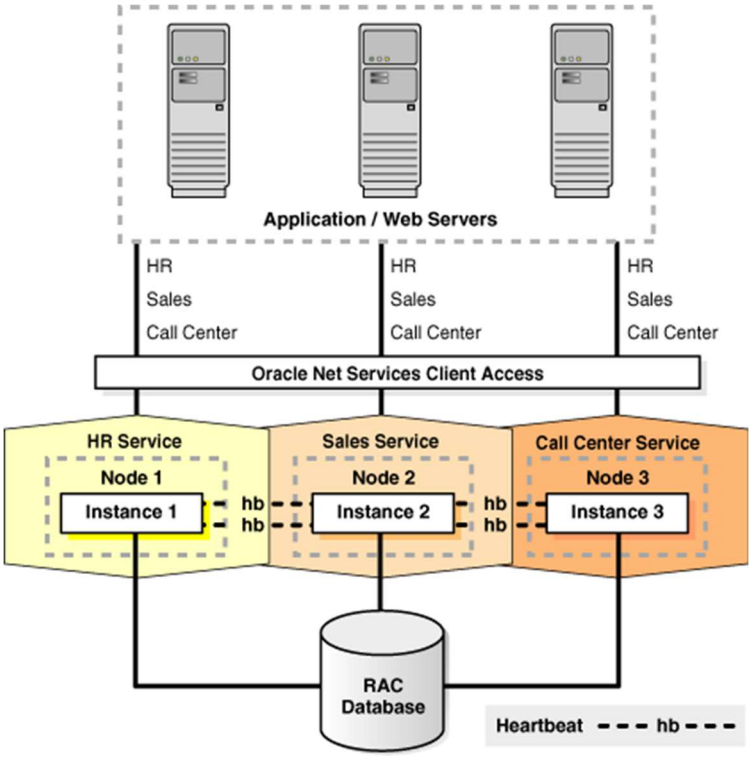


Figure 10 - RAC Database

2.4 – Runtime View

2.4.1 – Registration

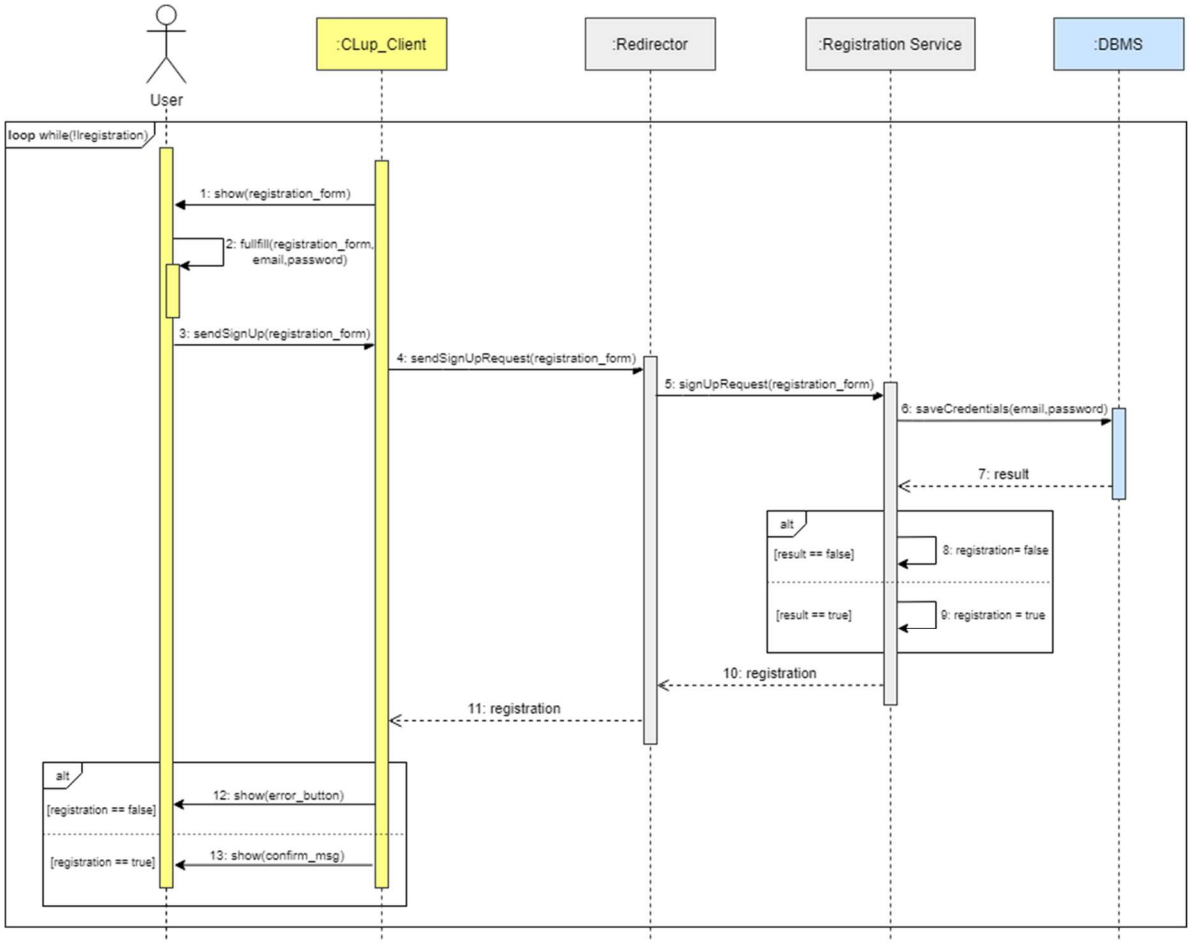


Figure 11 - Registration sequence diagram

In this sequence the process of registration is shown. The app shows the registration form to the user who will fill it with his/her email and password. Once done it the app will send it to the redirector that will forward it to the Registration Service, here the Registration Service will save his/her credentials in the database (checking that the email is not used yet). The result will be propagated back to the user and if it's positive a confirm message will be shown, otherwise an error will be shown.

2.4.2 – Login

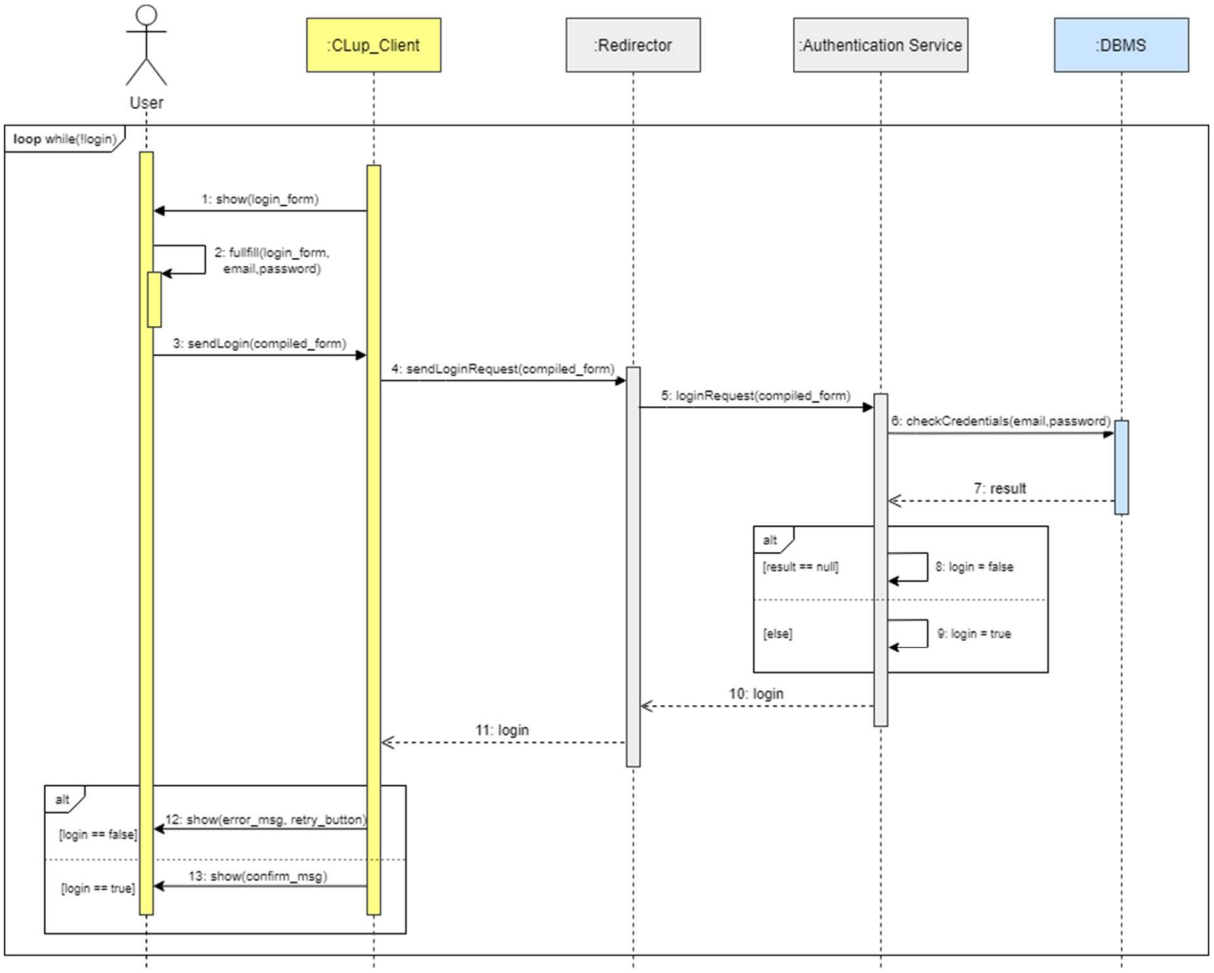


Figure 12 - Login sequence diagram

In this sequence the process of login is shown. The app shows the login form to the user who will fill it with his/her email and password. Once done it the app will send it to the redirector that will forward it to the Authentication Service, here the Authentication Service will check his/her credentials in the database. The result will be propagated back to the user and if it's positive a confirm message will be shown, otherwise an error message will be shown.

2.4.3 – LineUp Request

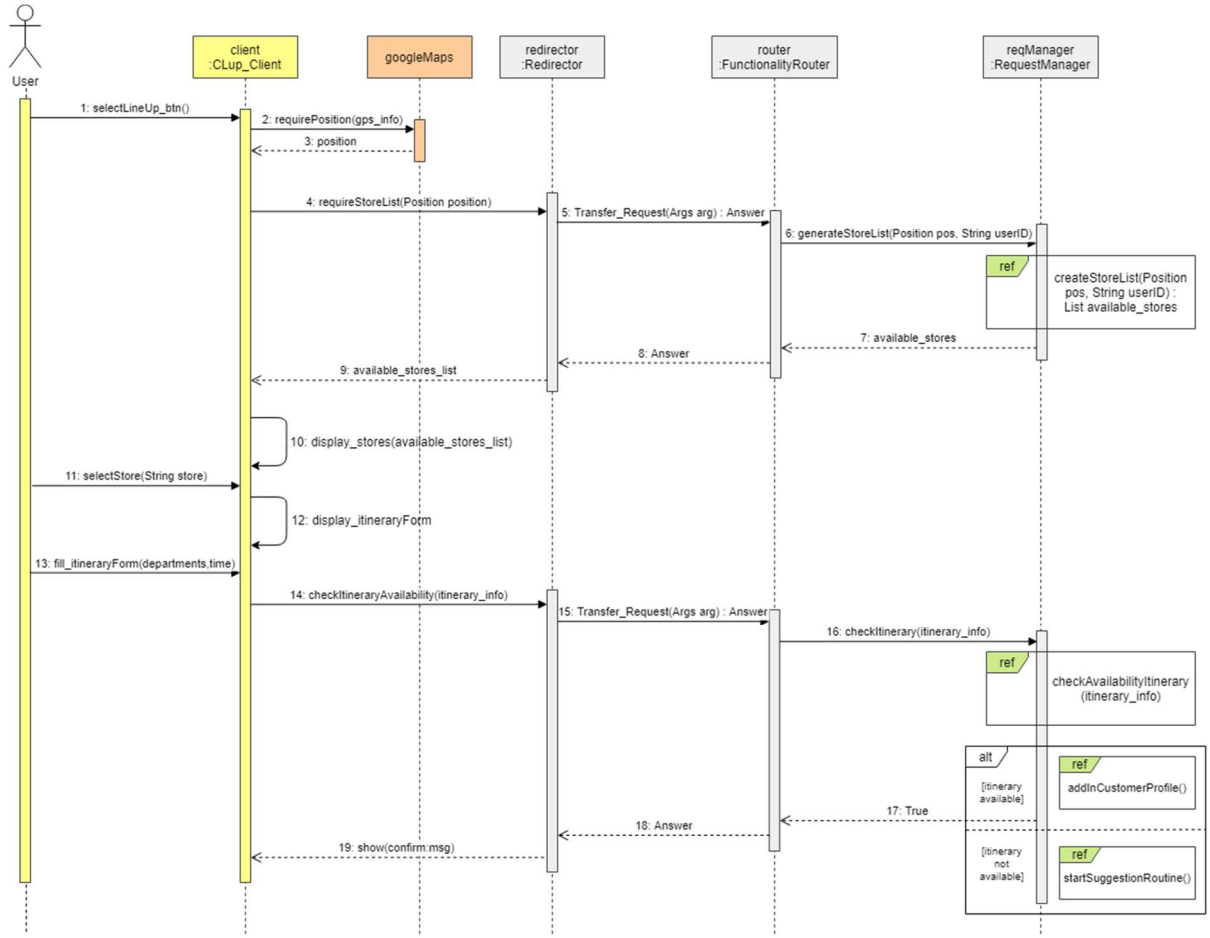


Figure 13 - LineUp Request sequence diagram

This sequence the sequence of LineUp Request, when a customer wants to be put in the queue of a store. In the first part CLup_Client retrieves the information of the physical position of the client asking to Google Maps service, then proceeds to send this information to the server part of the application, in particular to Request Manager which calculates, thanks to informations in the database, a list of the nearest store to the position given, it sends it to the CLup_Client which shows the list to the customer, the customer proceeds to select one store from the list and selecting an itinerary(also shown by CLup_Client), then client communicates the choice to RequestManager that checks if the store is available in the specified itinerary if so it adds the request on the active requests of the customer otherwise it proceeds with the SuggestionRoutine.

2.4.4 – Notify Leaving Home

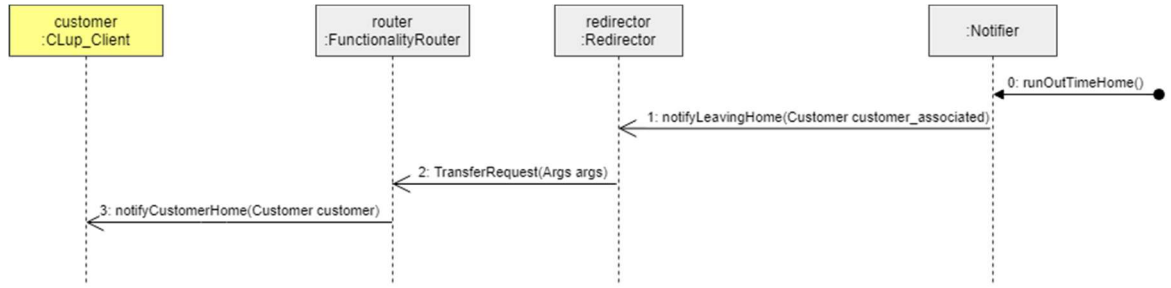


Figure 14 - Notify Leaving Home sequence diagram

The sequence of the notify about leaving home is rather simple. After an event that represents the run out of time by a specified timer for this use, the notify is sent to the customer. The messages are asynchronous because they don't presuppose an answer.

2.4.5 – Cancel Request

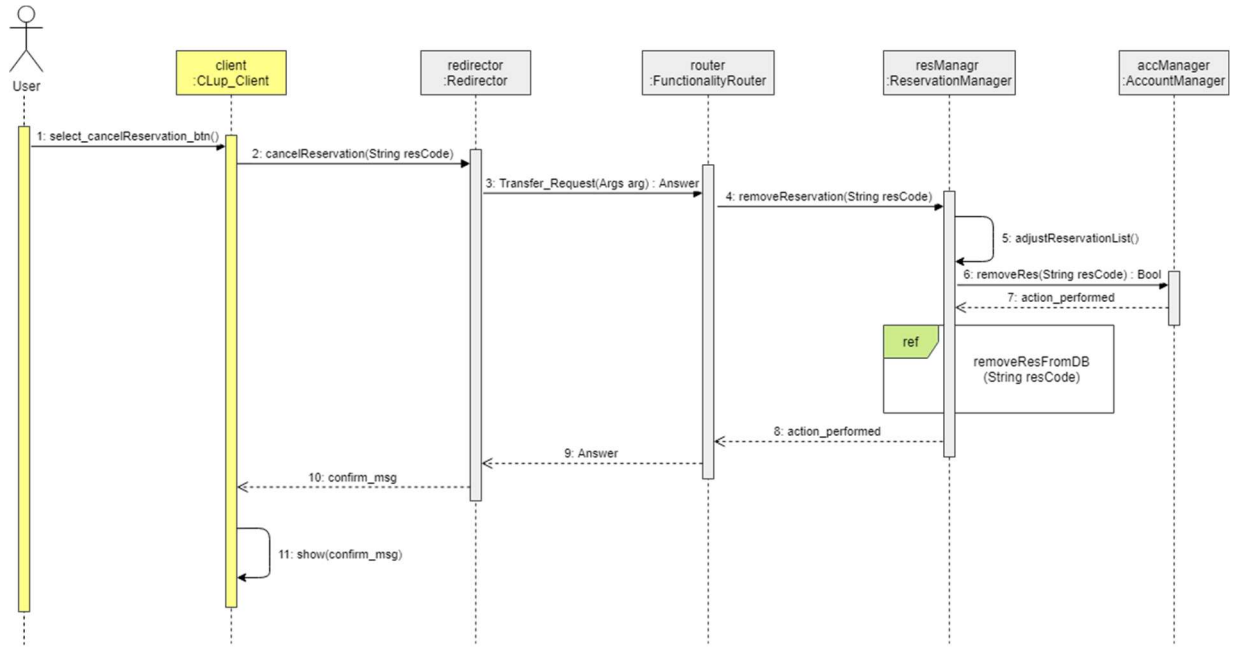


Figure 15 - Cancel Request

This sequence happens when the customer wants to cancel a reservation done, the cancelReservation request is sent from the CLup_Client to the Redirector that redirects it to the FunctionalityRouter thar routes it to the reservationManager, here happens the logic, the reservation is removed from the list in resManager that then sends a request to remove the reservation from the accountManager lists too, the account manager returns the result of the operation back through all the components

2.4.6 – Scan Code

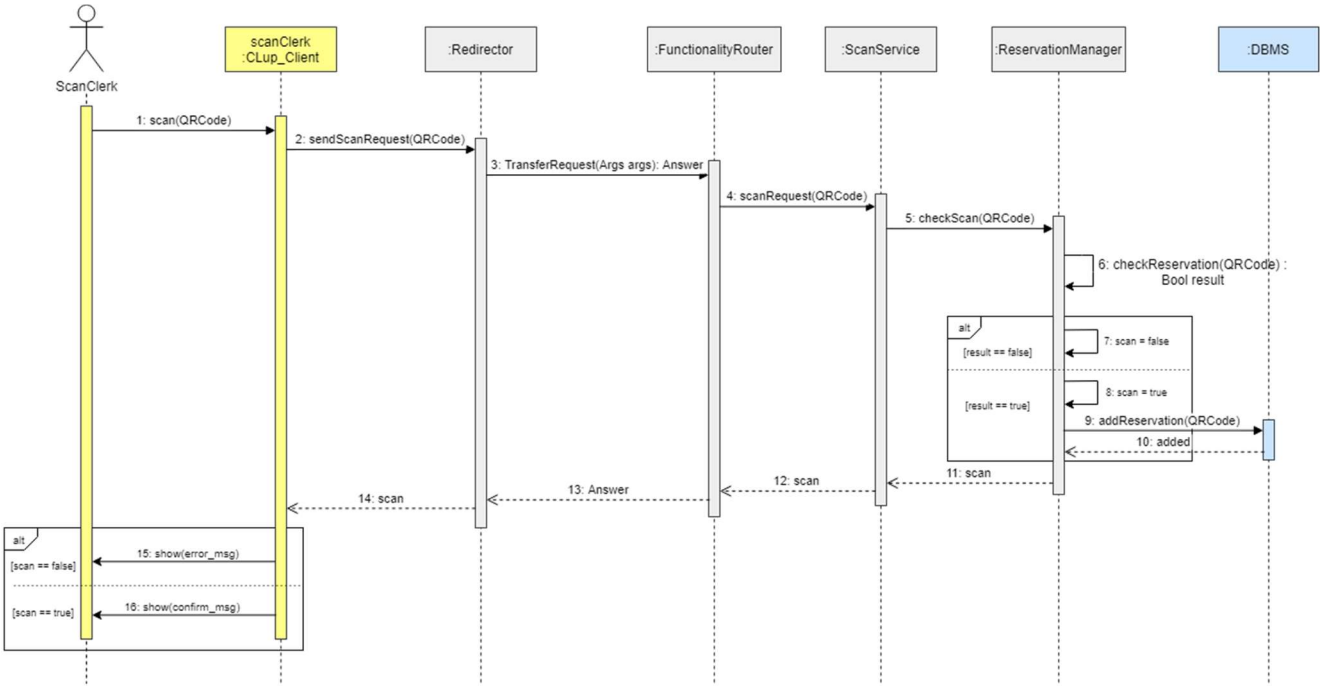


Figure 16 - Scan Code sequence diagram

In this sequence what happens when the scanClerk scans the code is shown. The scanning request is sent to the ScanService through the Redirector and the Functionality Router. Here the Reservation Manager is called to check if the reservation is valid (i.e. the time of entry is the same time of the scan) and if the answer is positive then the reservation is added in the database. The answer is propagated back to the scanClerk: if it's positive then a confirm message is shown, an error message otherwise.

2.4.7 – Suggestion other Store

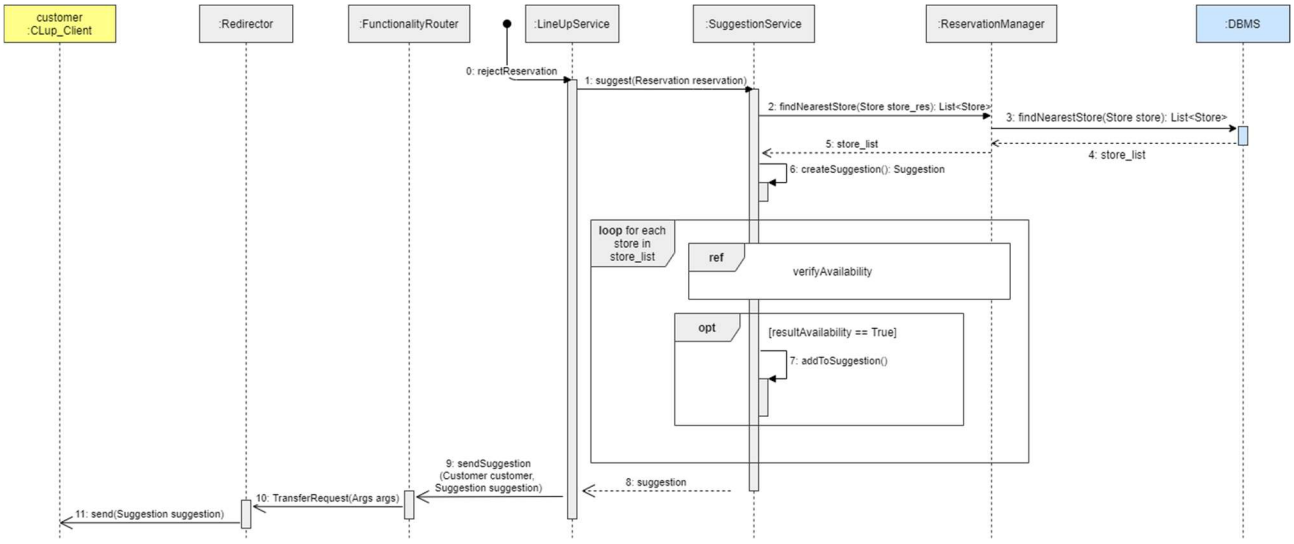


Figure 17 - Suggestion other Store sequence diagram

This diagram provides a focus on the suggestion service offered with a LineUp request refused. After the rejection, the operation begins. Initially, it finds the list of the nearest store through the use of Reservation Manager (and consequently the DBMS), verifies their availability for the type of request and then sends the suggestion to the customer. It could be empty if no store is available.

The last messages are asynchronous to avoid the representation of the activation parts (they would have started from the beginning without any initial messages). They would have been the return messages from the initial request.

The *SuggestionService* for Booking request are rather similar. It will provide, in addition, also a list of possible times in which the customer can book the same store the same day (if they exist).

2.4.8 – Store Manager

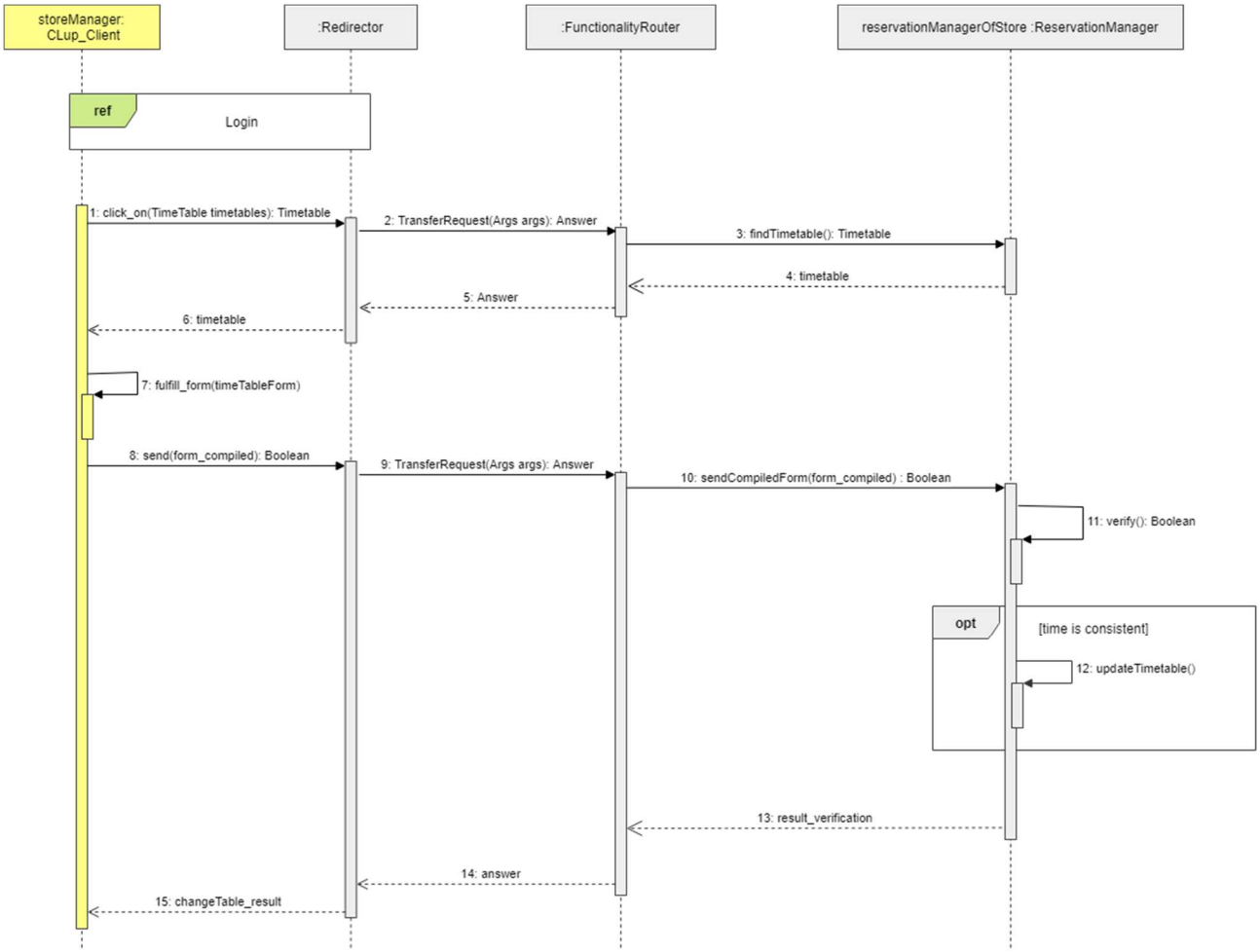


Figure 18 - Store Manager sequence diagram

This diagram offers the runtime view about the change of the timetable by the store manager. The structure is intuitive and not complex. After the login, the store manager will access to the timetable area, he/she will receive the current timetable. He/She will fulfil the form, the system will control if the time is consistent, in other words, for instance, if the opening time is less than the closure time. After the control, she/he will receive the result that specifies the operation success or the failure.

2.5 – Selected Architectural Style and Patterns

- **Model-View-Controller (MVC):**
  It has been selected to use MVC pattern in the version defined by Oracle, the software will be divided into three connected parts in order to separate the way information is represented internally in the Model part from the way it is represented and accepted from the user in the View, this pattern fits very well with a 3 tier architecture in which the business logic and the data management will be managed by the Model and the Controller components of the MVC, while the user interface (presentation tier) will be incorporated in the View.
  The main advantages of using this pattern are that we can avoid a high level of coupling between the various parts of the software and that we can create components independently of each other and simultaneous development is simplified.
- **Observer pattern:**
  This pattern is practically essential for the MVC because the controller is nothing else than a Observer of the model.
  It lets and object (the subject) notify his observers whenever its state changes, usually with an update method.
- **Three tier architecture:**
  It as been selected to use a Three tier architecture as architectural style for the software, it means that the entire software will be divide in:
  1- Presentation tier: it is practically the interface to the user in the client part of a client/server view
  2- Logic tier: this is the core of the application, when command, logical decisions and calculations are performed, it also connects the other two tiers
  3- Data tier: here is were the information are store and managed, normally in a database by a DBMS

  This type of architecture has the main advantage that every tier can be developed independently and that in case of future upgrades a single tier can be substituted without changing the other ones.

- **Client/Server with Thin Client:**
  A client/server architecture will be adopted, we decided to opt for a thin client model in which all the business logic will be in the server while the client will only have to perform requests and host the user interface because in this application nothing will ever happen offline but every action of the user must pass from a request to the server, the only action that the client can perform independently is retrieving his position through Google Maps and sending it to the server.

3 – User Interface Design

In this section the mockup of the app will be shown, as in the RASD:

Mockup 1 – Icon

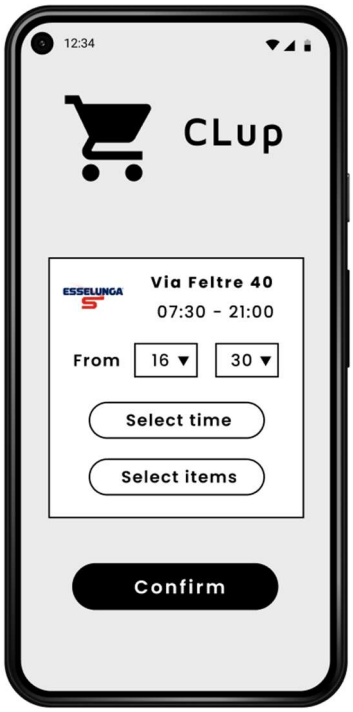
Mockup 2 – Login / Sign Up

Mockup 3 – Sign Up Form

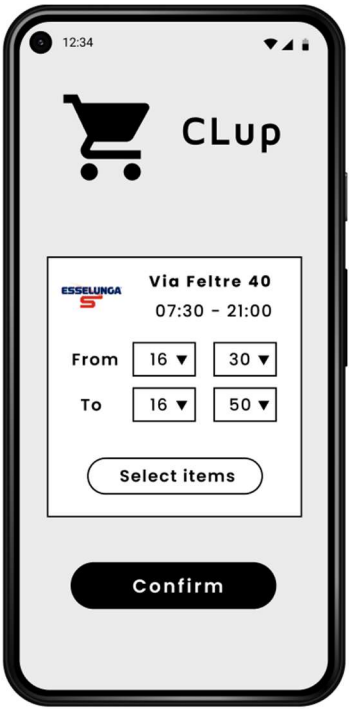
Mockup 4 – Login Form

Mockup 5 – Customer Home

Mockup 6 – List of Store

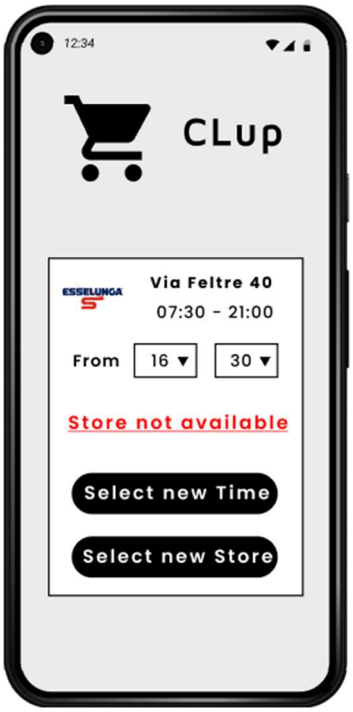


Mockup 7 – Reservation after selecting store



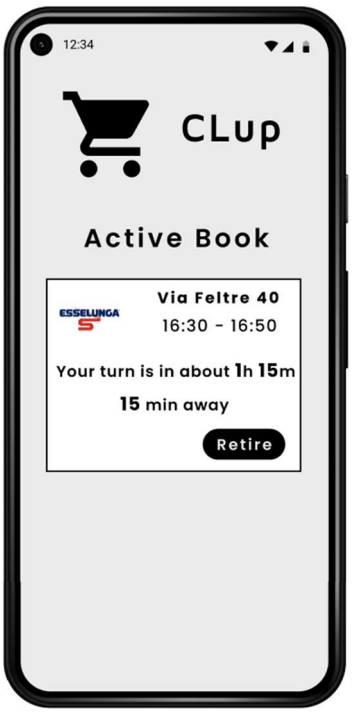
Mockup 8 – Reservation after selecting time of visit

Mockup 9 – List of Departments



Mockup 10 – Store not available

Mockup 11 – Customer Profile



Mockup 12 – Active Booking Info

Mockup 13 – Store Manager Home

Mockup 14 – Departments Live Info

4 – Requirements Traceability

The following table provides a mapping between goals and requirements defined in the RASD and system components illustrated in the DD.

- G1: The system must allow customers to line up

  Requirements: R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15

  Components:
  1. CLup_Client
  2. Redirector
  3. FunctionalityRouter
  4. RequestManager [LineUpService]

- G2: The system must allow store managers to regulate the maximum influx of people in the building

  Requirements: R16, R17, R18, R19, R20, R21, R22

  Components:
  1. CLup_Client
  2. Redirector
  3. FunctionalityRouter
  4. ReservationManager

- G3: The system must provide customers with an estimate of the waiting time

  Requirements: R23, R24

  Components:
  1. CLup_Client
  2. Redirector
  3. FunctionalityRouter
  4. ReservationManager

- G4: The system must manage customer entrances

  Requirements: R25, R26, R27, R28, R29, R30

  Components:
  1. CLup_Client
  2. Redirector
  3. FunctionalityRouter
  4. ScanService
  5. ReservationManager
  6. DBMS

- G5: The system must allow customer to book a visit in the store

  Requirements: R2, R7, R8, R9, R10, R11, R31, R32, R33, R34, R35, R36, R37, R38

  Components:
  1. CLup_Client
  2. Redirector
  3. FunctionalityRouter
  4. RequestManager [BookingService]

- G6: The system must alert the customers when they have to leave to arrive in time at the store

  Requirements: R39, R40, R41

  Components:
  1. AccountManager [Notifier]
  2. Redirector
  3. FunctionalityRouter
  4. CLup_Client

This part is a remind of all the requirements described in the RASD:

R1: The application allows customers to make a reservation only if it is possible to enter the store

R2: The system generates a unique code for each reservation

R3: The system frees customers from the queue if they're late more than 5 minutes

R4: The system must show the predicted waiting time of the customer

R5: The customer who lines up is inserted in the queue of the store

R6: If a customer has already lined up in a store and he/she hasn't left the queue yet, he/she can't line up again in it

R7: The system must allow customers to specify the duration of their visit

R8: The system must allow customers to specify the departments of the items they want to buy

R9: The system provides the customer a list of the stores opened at that time if the preferred one is unavailable

R10: The system must suggest different stores of the same or even different chain if the preferred one isn't available

R11: The system must allow customers to cancel their reservation before they enter the store

R12: The system confirms the customer the acceptance of the asked reservation

R13: In case of confirmed reservation the system must insert the customer in the queue of the selected store

R14: The system must reject line up requests if the stores chosen are not available

R15: The system must permit only one line up a time for the same store and the same customer

R16: The application limit access to the store if the number of people is equal to the maximum possible

R17: The system must allow to monitor the entrances

R18: The system must allow to monitor the leavings

R19: The system must allow store managers to set the maximum number that can enter the store

R20: The system allows store manager to define the zones of the store

R21: The system allows store manager to define the maximum number of people for every zone in the store

R22: The system must provide the store manager the number of customers for every zone at the moment

R23: The system must keep track of the average time of permanence of customers in the store

R24: The system sends a notification if the new waiting time is less than 20 minutes in case of missed customers

R25: After the scan of the unique code the system removes the customer from the queue

R26: After the scan of the unique code the system inserts the customer into the store list of active customers

R27: The system must send a notification to the customer when it's her/his time to enter.

R28: After an active customer leaves the store (through the re-scanning of the same code) , the same customer is removed from the active customer of the store

R29: When there is a free space in the active customers, the system must decide which is the next customer to enter

R30: If there is a free space available in the store and no online customer can reach the store in time, the first hand out customer in the queue can enter the store

R31: The system permits the customer to select which store he wants to book a visit

R32: The system permits the customer to select when he/she will go to the store

R33: The system confirms the customer the acceptance of the asked reservation

R34: In case of confirmed reservation the system must insert the customer in the queue of the selected store

R35: The system includes suggestions of alternative slots to customers if the selected time isn't available

R36: The system must allow customers to select a preferred store

R37: The system must reject bookings in which the time chosen is one not available in the store

R38: The system must reject bookings if their times for the same customer intersect each other

R39: The system must calculate the distance between the customer and the store

R40: The system must acquire the customer position

R41: The system must send a notification if the waiting time of the queue is less than 5 minutes plus the arrival time to the store from their position

5 – Implementation, Integration and Test Plan

5.1 – Overview

In this part an implementation plan is offered in according with the integration strategy. After an analysis of the structure of the project, they're both developed in order to avoid incongruency and wasted time. They're based on the need to have more time to test the critical key part of the system. Moreover, some considerations about testing will be provided.

5.2 – Implementation Plan

The entire system (with its relative subsystem) is implemented and integrated adopting a bottom-up approach. This method allows the developers to start the integration in parallel with the implementation, gaining different benefits, such as:

- Finding errors in the earlier step, avoiding the increasing of the cost to fix them
- Focusing on the dependency between each component
- Having more time to control and track the critical parts during the development of the entire project

Even if the approach guarantees a theoretical stability of the implementation, the decision of which component comes first is crucial. The order must be based on the idea in the third point of the previous list. The developers must have time to check and track the critical part. Therefore, the structure of the implementation plan must have coherent with what written before and need a deeper analysis of the system.

The external components won't be implemented because provided by third part, such as: *GoogleMaps, Printer* and *Monitor*.

The main functions are provided by *RequestManager* and *ReservationManager*. The first one is able to create a request, to delete an existing one and to verify the feasibility of the one already created. The second one organizes the queue of every store and manages the entrances and the leavings. They're the critical part of the system. Without these two components all system can't execute as described.

After them, in the application the key independent component is obviously the *DBMS* which has the function to connect the system with the database in order to store all the relevant data. In parallel also the *Account Manager* can be implemented because it's responsible only for the notifier and for tracking the main parameters of every customer. After these two, the plan suggests the implementation of *ReservationManager*. In the second phase, the implementation will already reach one of the two most relevant components.

To permits also the development of *RequestManager*, the system needs *SuggestionService* to complete all the service that can be offered by it. After that, the second important component can be realised.

After them, the developers can focus on the login/registration structure with the *RegistrationAndLogin _Manager* subsystem. The last two parts focus on the communication part and the client side. The communication system is offered by *FunctionalityRouter* and *Redirector*. At this point the server side is totally completed and the teams can test it with all functionality implemented.

Finally, the *CLup_Client* will be implemented.

To summarize, the order will be this:

- *DBMS* in parallel with *AccountManager*
- *ReservationManager*
- *SuggestionService*
- *RequestManager*
- *ScanService* in parallel with *RegistrationAndLogin_Manager*
- *Router*

- *Redirector*
- *CLup_Client*

5.3 – Integration Strategy

To implement and test the different components a bottom-up approach will be used, to do so Driver is defined as the components that are still under implementation, this will be useful for the tests:

1. Firstly the components DBMS and AccountManager are implemented because none of them is directly dependent from any class and other components rely on them, that's why they are implemented and tested before everything else, in this step Google Maps will also be integrated, it doesn't need to be implemented and unit tested because it is a service given by a trusted provider.

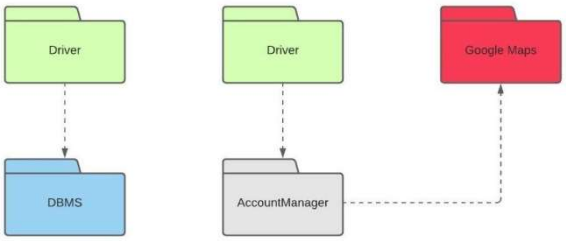


Figure 18 – Integration after step 1

2. Then ReservationManager will be implemented because it is a key component of the system, many other components rely on it so it should be integrated immediately after DBMS and AccountManager, at this step also Monitor will be integrated, like Google Maps the monitor doesn't need to be unit tested because it is an external service.

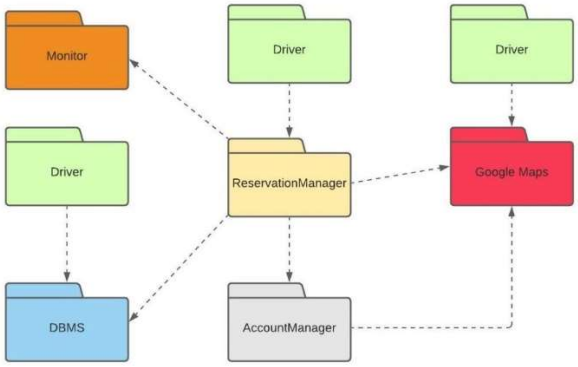


Figure 19 – Integration after step 2

3. At this point suggestion service will be implemented, which is the only component independent from any other non integrated component.

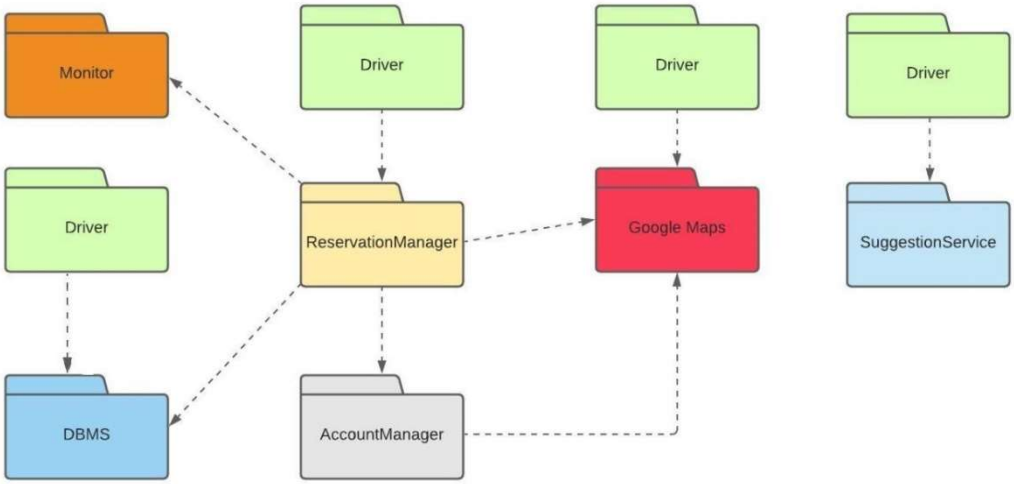


Figure 20 – Integration after step 3

4. Then Printer will be integrated in the system, like Google Maps and Monitor this component doesn't need to be unit tested because it is an external service by a trusted provider, immediately after Printer all the components on which RequestManager relies will be integrated so it will be implemented and unit tested.

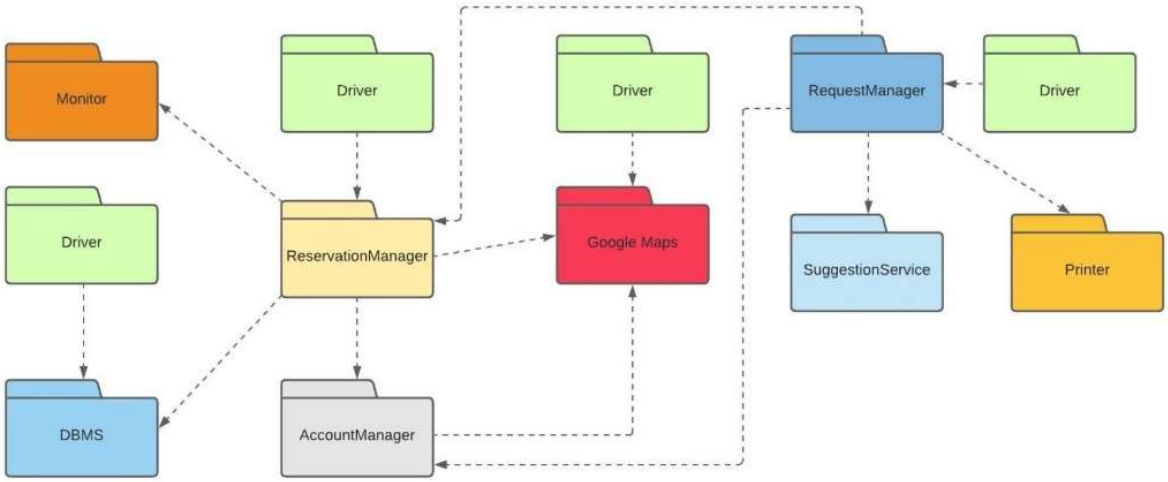


Figure 21 – Integration after step 4

5. Then ScanService and RegistrationAndLogin_Manager will be implemented and unit tested in parallel because they are not directly dependent.

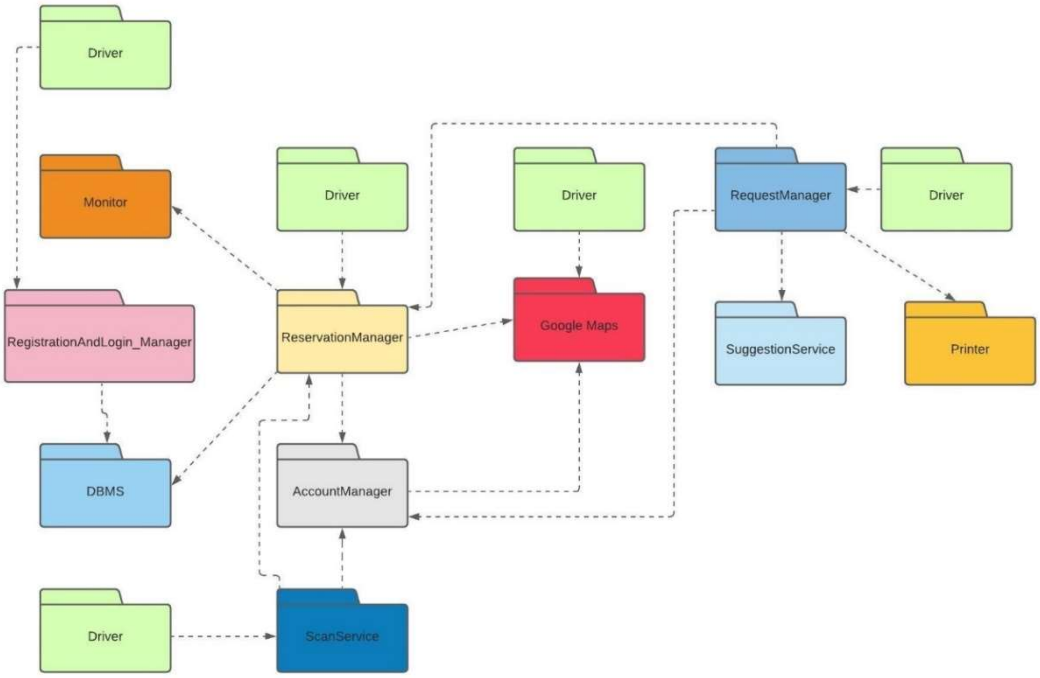


Figure 22 – Integration after step 5

6. At the end FunctionalityRouter will be implemented and unit tested, directly followed by Redirector, this two components will take the place of Driver.

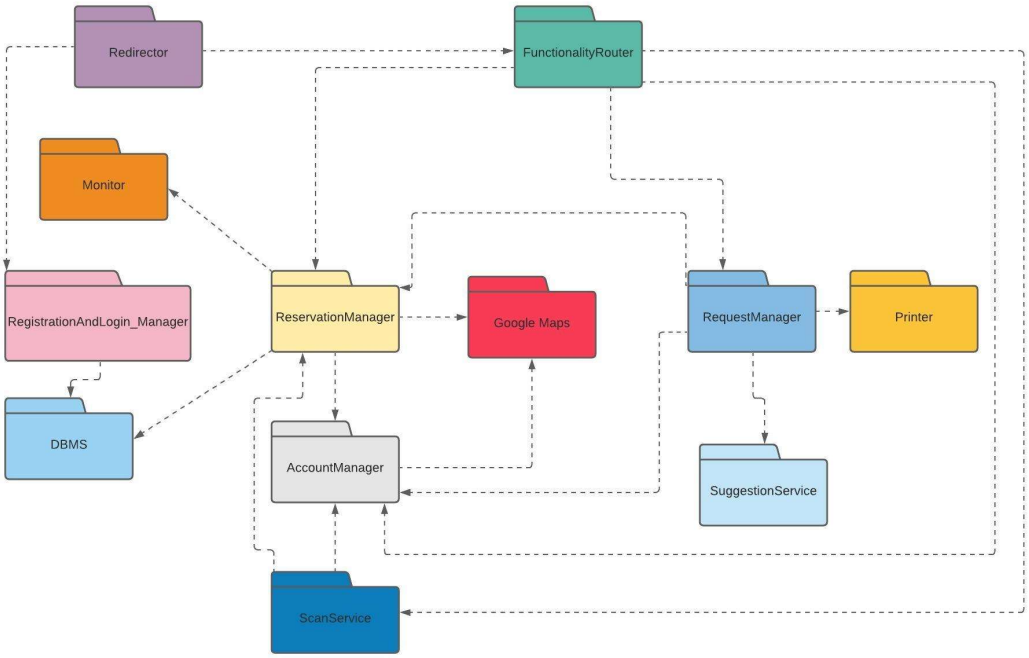


Figure 23 – Integration after step 6

5.4 – System Testing

Once the System is completely integrated, it must be tested as a whole to verify that functional and non-functional requirements are satisfied. The system testing is the means by which it is possible to do so. Moreover, the testing environment should be as close as possible to the production environment.

The system testing can be divided in more types:

- **Functional testing**: verifies if the application satisfies the functional requirements described in the RASD.
- **Performance testing**: identifies bottlenecks affecting response time, utilization, throughput and establishes a performance baseline and possibly compares it with different versions of the same product or a different competitive product (benchmarking). Thanks to this it is possible to identify the presence of inefficient algorithms, query optimization possibilities or hardware/network issues.
- **Load testing**: exposes bugs such as memory leaks, mismanagement of memory, buffer overflows and identifies upper limits of components.
- **Stress testing**: makes sure that the system recovers gracefully after failure.

5.5 – Additional Specification on Testing

The developers must produce functional test cases together with the code. It is expected that at least 85% of the code belonging to the model (referring to MVC pattern) will be covered with test cases specially the most critical parts of it. A fully coverage (or at least 85%) of unit testing conduct by developers itself helps to find problems early.

In specific test case the activity should consist in systematically testing activities concerning the characteristics and the structure of the code (white-box testing). Developers must choose inputs for the units and determine if the output is appropriate for every module.

An important thing to do is to comment the code. Indeed, the developers are strongly invited to write well commented code. Semi-formal notations are also required to be written in order to assure a fully description of the code.

6 – Effort Spent

Simone Bianchera

Chapter	Hours
1 – Introduction	4h
2 – Architectural Design	11h
3 – User Interface Design	0h
4 – Requirements Traceability	1h
5 – Integration, Test and Plan	5h

Niccolò Borrelli

Chapter	Hours
1 – Introduction	4h
2 – Architectural Design	7h
3 – User Interface Design	7h
4 – Requirements Traceability	2h
5 – Integration, Test and Plan	3h

Rei Barjami

Chapter	Hours
1 – Introduction	4h
2 – Architectural Design	7h
3 – User Interface Design	0h
4 – Requirements Traceability	2h
5 – Integration, Test and Plan	8h