POLITECNICO
MILANO 1863

Design Document of CLup

Version 2.0 - 11 January 2021

Pasquale Occhinegro
Giampiero Repole
Andrea Zanetti

Politecnico di Milano - A.A. 2020/2021

1. Introduction

1.1 Purpose

The purpose of this document is giving a technical analysis and a more in-depth description of the project presented in the RASD document. A detailed explanation of how the product functions and the user interfaces are implemented is given in the following sections. Short explanations will be given for all the options chosen at every time it is needed.

1.2 Scope

CLup (Customers Line up) is an application that allows people to safely approach an essential task such as grocery shopping during emergency times like the current pandemic.
People can sign-up on the app and provide their address to see a list of nearby stores from which they can choose one to line-up in a digital queue handled by the system.
When queuing, the system chooses the closest interval of time that satisfies the safety conditions of the store, and tells the Customer the estimated time of arrival and that remaining until his turn.
The Customer can also book the visit, specifying the store, their estimated time of staying and a general idea of the items they want to buy.
In this case the system sends the Customer a list of slots to choose from.
The Customer can always cancel a visit previously booked.
The system also memorizes the booking activity and during a new instance from the Customer the app will suggest the same time slots that were selected in his previous bookings.
All Customers are provided with a QR code, when they book a visit or line-up, that must be used at the store entrance to keep in check the number of people inside.
The Customers can also line-up in the virtual queue at the entrance of the store, where a physical QR code will be printed along with the estimated time remaining until his turn.
A Store Manager can submit their store in the app, specifying the number of people allowed in each department, and then monitor the number of people inside the store thanks to the QR code and the estimated time of staying.

1.3 Definitions, acronyms, Abbreviations

1.3.1 Definitions

Customer	A person who uses our app to queue up or book a visit
Store Manager	A person that uses our app to check the number of people in his store and manages the entrances
Map Provider	A third-party company that provides his services

1.3.2 Acronyms

RASD	Requirements Analysis and Specification Document
DD	Design Document
GPS	Global Positioning System
DB	DataBase
DBMS	DataBase Management System
REST	REpresentational State Transfer
API	Application Programming Interface
FCM	Firebase Cloud Messaging

1.3.3 Abbreviations

Rn	Requirement number n
Gn	Goal number n

1.4 Revision History

- Version 1.0: first release
- Version 2.0: fixed typos, chapter 5 page layout

1.5 Reference Documents

- P. Occhinegro, G. Repole and A. Zanetti, Requirements Analysis and Specification Document of CLup, Milan, 2020.

- Slides of the lectures

1.6 Document Structure

This document is divided in 7 chapters, briefly described below:

- **Chapter 1**: a summary of the document is presented here, specifically it describes what this document is about and how the tasks and problems are going to be tackled.

- **Chapter 2**: architectural design choices are presented, including components, interfaces and technologies used to develop the application. Then it describes more in detail the architectural choices, including functions and interfaces related to the processes of the application where they are used.

- **Chapter 3**: lists some mock-ups of the General User Interface associated with the corresponding feature shown.

- **Chapter 4**: describes the connection between the RASD and the DD, where each requirement is linked to the goals previously listed with the elements which form the architecture of the application

- **Chapter 5**: lays out the plan to develop the components. It is divided into two sections: implementation and integration. Finally, it includes the testing strategy.

- **Chapter 6**: shows the effort spent by each member of the group.

- **Chapter 7**: includes the reference documents.

2. Architectural Design

2.1 Overview

In this section we describe the architecture of the CLup application from a high-level perspective.
The application is thought to implement a multi-tiered client-server architecture according to the various components that make up the system.
In particular it's organized in a more generic three-tiered architecture:

- **Presentation Layer (P):** manages interaction between Users and real application offering a GUI which shows data sent from lower layers and takes input from Users;

- **Business Logic Layer (B):** the core of the application, that through a middleware (RequestHandler) manages the inputs of the Users and data in the lowest layer following the logic of the application;

- **Data Access Layer (D):** DBMS server that contains all the data useful to the correct functioning of the application.

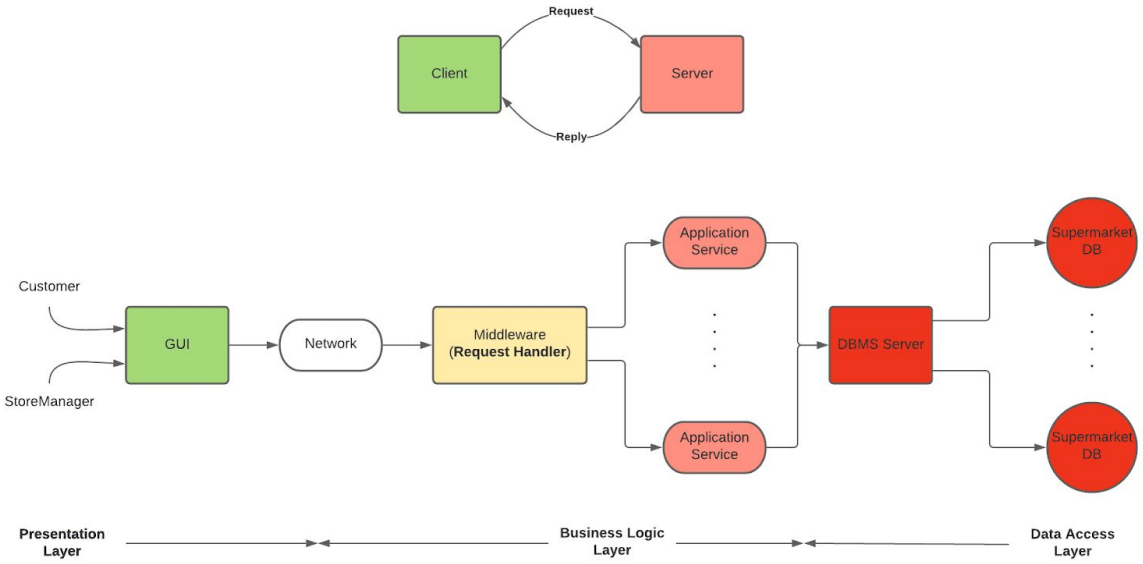


Figure 1: High-Level multi-tier architecture

Each layer is implemented on different hardware in order to have a physical and a logic distinction among them.
This architecture offers advantages to the development and to the maintenance of the application like scalability, performance, availability and also an improved speed of development.

2.1.1 High Level Components

As described above, the application is structured in three-tier and the process begins in the presentation layer when a Customer or a Store Manager interacts with the Client, corresponding to their smartphone, that sends an HTTP request to the Server to invoke a CLup's service.
On the Server side, a RESTful interface takes care of retrieving Client requests and selecting the component that implements the service corresponding to the functionality selected by the user:

- **Basic Service:**
    - A Customer wants to virtually line-up to enter in a selected Supermarket: the Server checks on the Supermarket's DB the last available position in the queue, if it is not already full otherwise warns the Client, and calculates the estimated time until his turn.
      Then the Server stores the operation and sends a ticket to the Client with all information to enter the store.
    - A Store Manager wants to check the accesses to the store and consequently, manage the capacity of the supermarket and the various departments if needed: the Server retrieves the requested data from the Supermarket's DB, sends them to the Client.
      Than, if the Store Manager wants to regulate the influx, the Server takes the inputs and update the DB
- **Map Service:**
    - A Customer wants to view the Supermarkets near his position: the Server asks the Map Provider the requested information and shows them to the Client.
- **Advanced Services:**
    - A customer wants to book a visit to a selected Supermarket: the Server asks the Client the departments in which the Customer intends to buy and number of slots that he wants to occupy, shows also slots occupied in the last booked visit in case of long term Customer.
      After that it retrieves the list of time slots from the Supermmarket's DB and asks the Customer to choose one of them.

If the Customer accepts the options, the operation is stored on the DB and the Server sends a ticket with all the information required.

- The Customer selects a not available time slot and chooses to be notified when the slot is again available: the Server periodically checks on the DB the status of the selected time slot and sends a notification to the Customer when the condition is verified.

- The Customer selects a not available time slot and he chooses to try in another Supermarket: the Server

The Client's device is a smartphone or a tablet and the interface differs for the Customers and the Store Managers only when they are logged in.

2.2 Component View

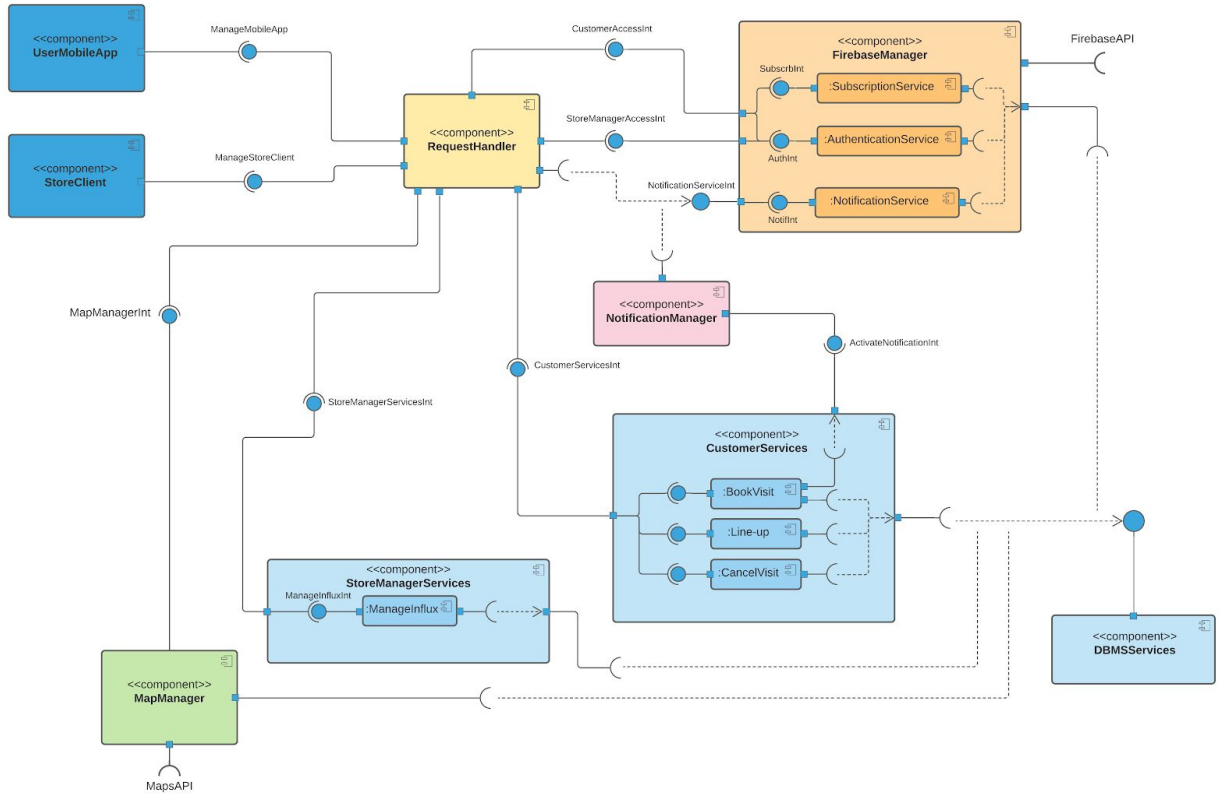


Figure 2: Component Diagram

The Component diagram above, focuses on how the internal components of the Application Server interact with each other and the external components. External components of Application Server (Google Maps, Firebase) have been displayed in an approximated way since they are already built and deployed.

- **StoreManagerServices**: this component lets the Store Manager regulate the influx of Customers entering the store. This feature is activated only in case the store capacity is soon to be exceeded.

- **NotificationManager**: this component manages the notification activation service available for the Client and the incoming notifications to send from the Server to the Client if they have been previously enabled.

- **CustomerServices**: this component is formed by three subparts:
    - **BookAVisit**: this component handles the request to book a visit in a store. It checks that the reservation has been correctly done and sends all the information to the **DBMSServices** to be stored.
    - **Line-up**: this component collects the requests from Customers to virtually line-up and calculates the position in line in which the Customer will be inserted. Once the request has been approved from the server all the details are sent to the **DBMSServices** for storage.
    - **CancelAVisit**: this component lets a Customer cancel a previously booked visit to a grocery store. If the cancellation process ends successfully all the information is updated through the **DBMSServices**.

- **RequestHandler**: this component is a RESTful interface that dispatches all the requests coming from Customers and Store Managers to the core of the application routed through the appropriate component that manages the requested feature. It is also responsible to send back data and responses to the Customer or Store Manager.

- **DBMSServices**: this component allows all the other components to interact with the database. It is provided with store, retrieve and update functions that will operate on data from the database when requested from different actors.

External components connected to the Application Server include:

- **MapManager**: this component provides a visual interface to display the map in which the Customers will choose the grocery store.

- **FirebaseManager**: this component provides authentication APIs to let Customers and Store Managers create an account the first time they access the App and to login the following times. Since the authentication process relies on cloud services offered by

Firebase, the App provider doesn't need to structure an internal system just to keep track of registered and logged users.
It also provides APIs to send notifications to Customers that have them activated on their account.

2.2.1 Additional Specifications

The configuration adopted for the Client is the one of Thin Client, where all the application logic is included in the Application Server. This is because for this type of application it is not needed to do any offline check since the Customer or Store Manager must be always connected to the Internet in order to perform actions that will interact with the Application Server.

2.3 Deployment View

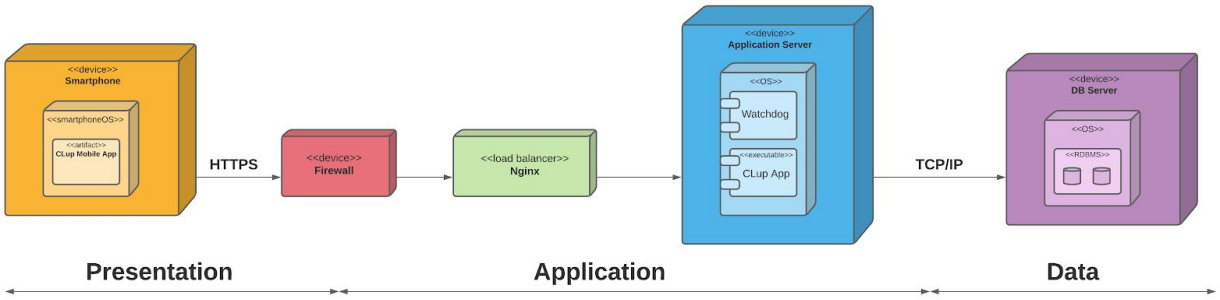


Figure 3: Deployment Diagram

In the Deployment diagram above are shown the most important components. It is also shown how the system (system software) will be physically distributed on the hardware. Systems such as Google Maps and Firebase are not shown here because they are already built and deployed. The Application is divided in the Presentation, Application and Data layers, and these three layers respectively contain:

- **Smartphone**: a smartphone is used by Customers to virtually line-up or to book a visit for a store. The Store Manager uses it to regulate the influx of people entering the store.

- **Firewall**: protects access to internal parts of the network against external attacks where the Application Server and DBMS are located.

- **Load balancer**: distributes the workload of requests to the server across multiple servers in order to increase capacity and avoid overload.

- **Application Server**: includes all of the application logic, since the Client is a Thin Client. Handles requests from the Client and sends responses for the services included in the App. The Application Server also includes a Watchdog to monitor component instances. In the event of failure it will manage their replacement.

- **Database Management System (DBMS)**: here data is stored safely in the Database Server which is executed as a relational DBMS (RDBMS). In this way the database is structured in such a way to easily link information from different tables.

2.4 Runtime View

In this section the runtime views of some of the most important functions of the application are shown.
Specifically, interaction between Customers and CLup, as well as between a Store Manager and the app, are visually described through their respective sequence diagrams.

2.4.1 Map

- **Providing Map:** since both the lining-up and the book a visit functions need the Customer to first choose a store in the map section, a specific sequence diagram was made in order to avoid repetitions.
  The main component is the Map Provider, that after being invoked asks the DBMS information about store locations, then renders the map and sends it to the Mobile App.

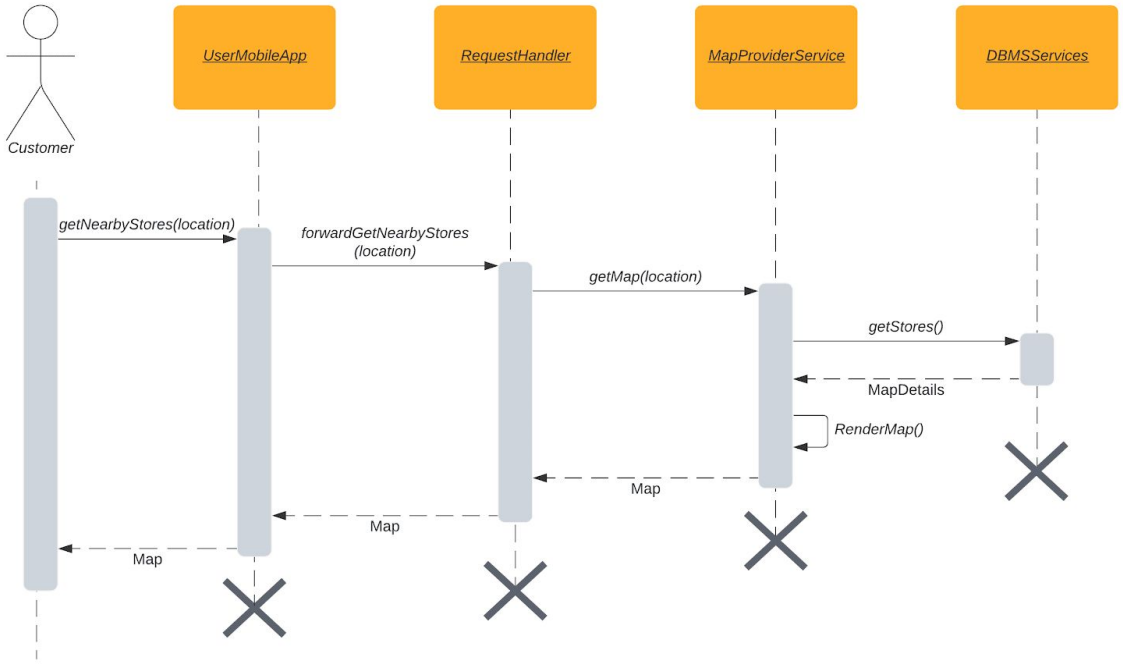


Figure 4: Provide Map View

2.4.2 Customer

- **Lining-up:** When a store is chosen, the Customer can select between the line up or the book a visit function, selecting the former sends a request up to the CustomerServices, which first asks the DBMS the slots that specific store has, and then checks if a slot is still available.
  If it is not, a message to the app is sent asking the Customer to choose another store.
  If it is, the CustomerServices updates the DBMS, locking the slot, and sends the QRCode to the MobileApp, that invokes the MapProviderService in order to calculate the estimated time of arrival thanks to the location of the Customer and the store.
  Finally, the MobileApp sends the Customer all that is needed.

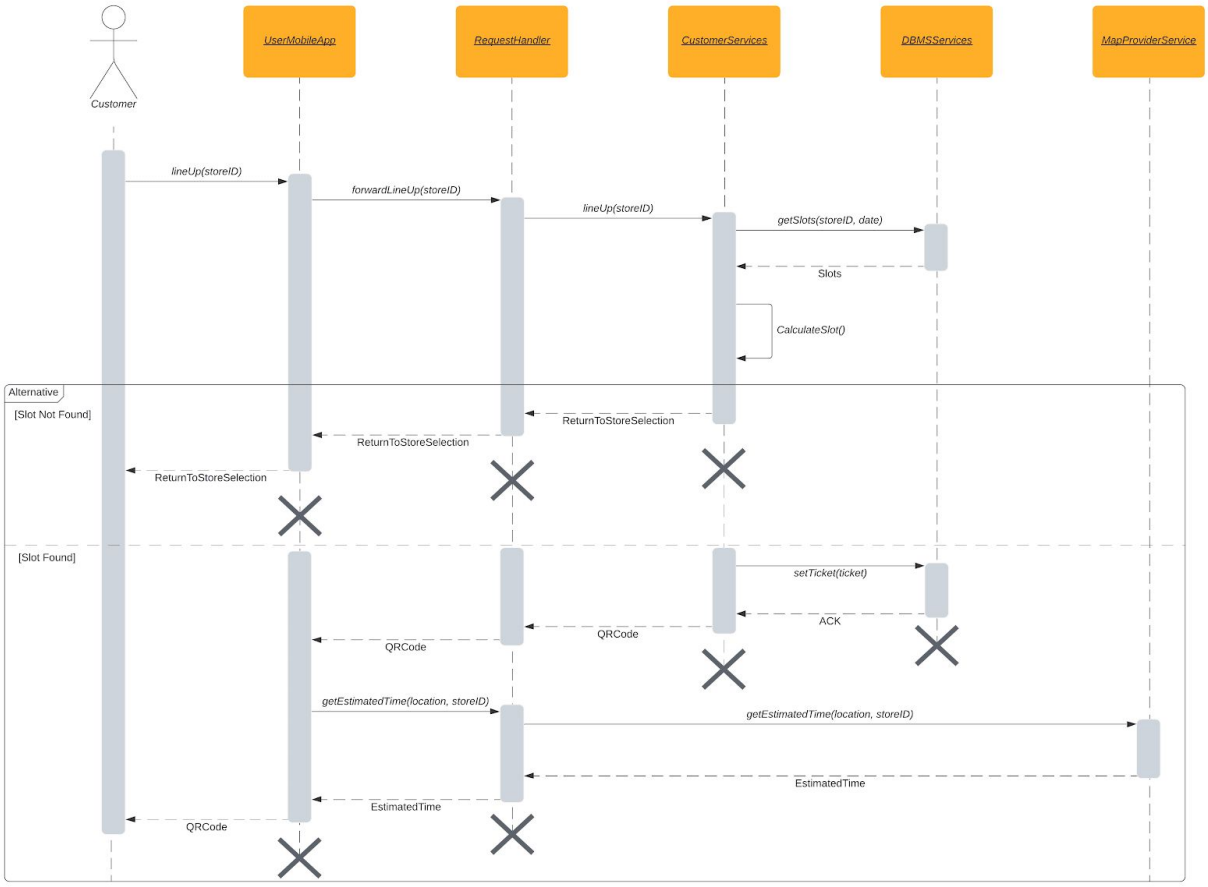


Figure 5: Lining Up View

- **Book a visit:** After the store selection the Customer can choose the book a visit function.
  If they are a long term customer they can request the slot numbers used in their last visit, retrieved in the DBMS.
  The Customer then confirms the number of slots and adds the departments they want to visit, and sends the request to the CustomerService, which provides the Customer the slot list of that day.
  After a slot is chosen, the CustomerService checks if it is still available.
  If it is not, it asks the Customer which solution they want to use between the notification or selecting another store, choosing the former will result in a request for the NotificationManager by the CustomerService, which will then inform the Customer.
  If the slot chosen is free, the DBMS is updated and the QRcode is sent to the Customer.

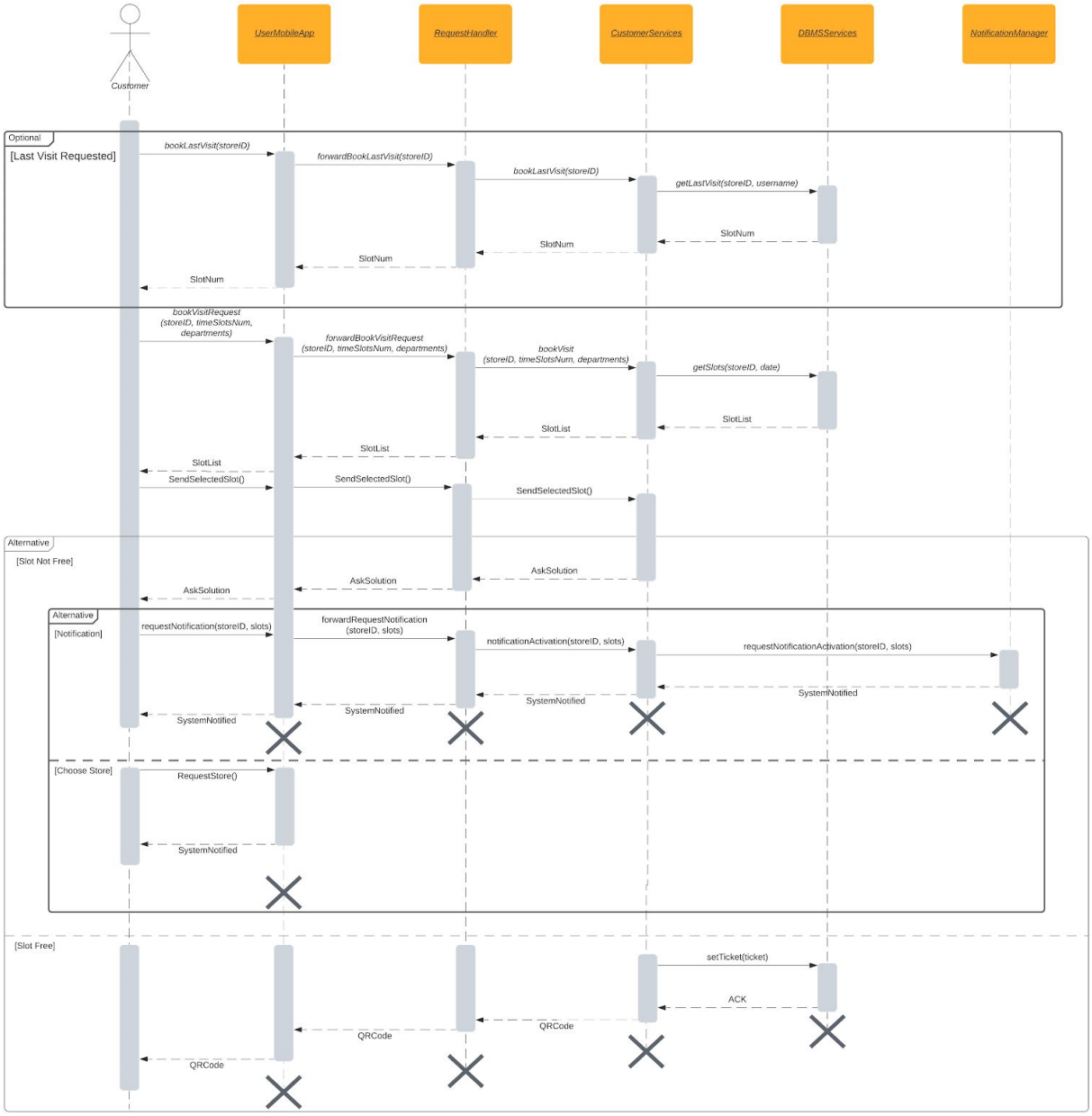


Figure 6: Book a Visit View

- **Cancel a visit:** The Customer can always see his booked visit with a request to the CustomerService, which gets the list from the DBMS.
  From here they can choose a visit from the list and after a second confirmation asked by the MobileApp the CustomerService updates the DBMS deleting that visit and freeing the slot.

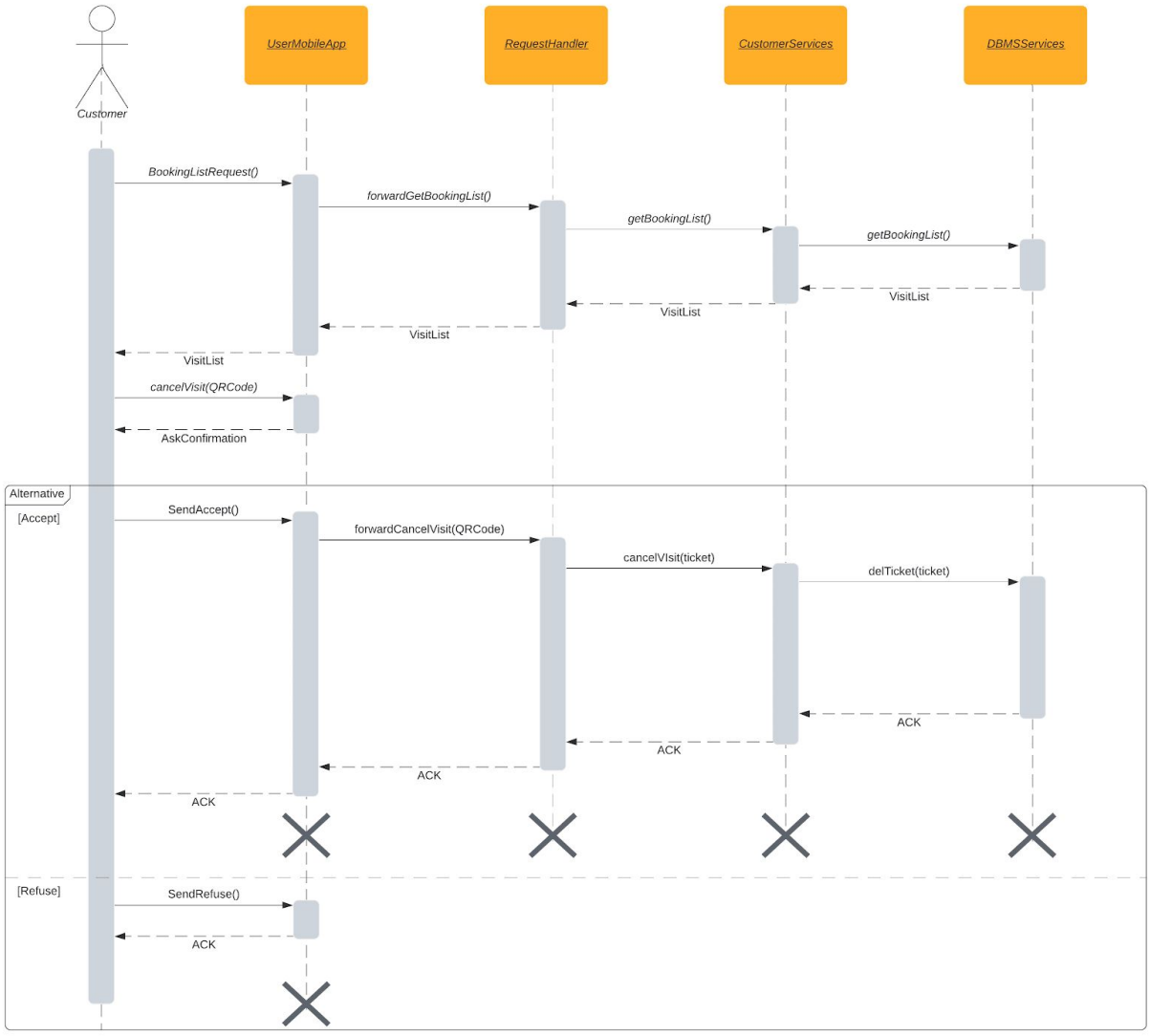


Figure 7: Cancel Visit View

2.4.3 Store Manager

- **Check Influx:** From the application the Store Manager can both check the number of accesses inside his store by sending the request to the StoreManagerService, or can change the number of people admitted inside his store or a specific department, in which case the StoreManagerService updates the number inside the DBMS.

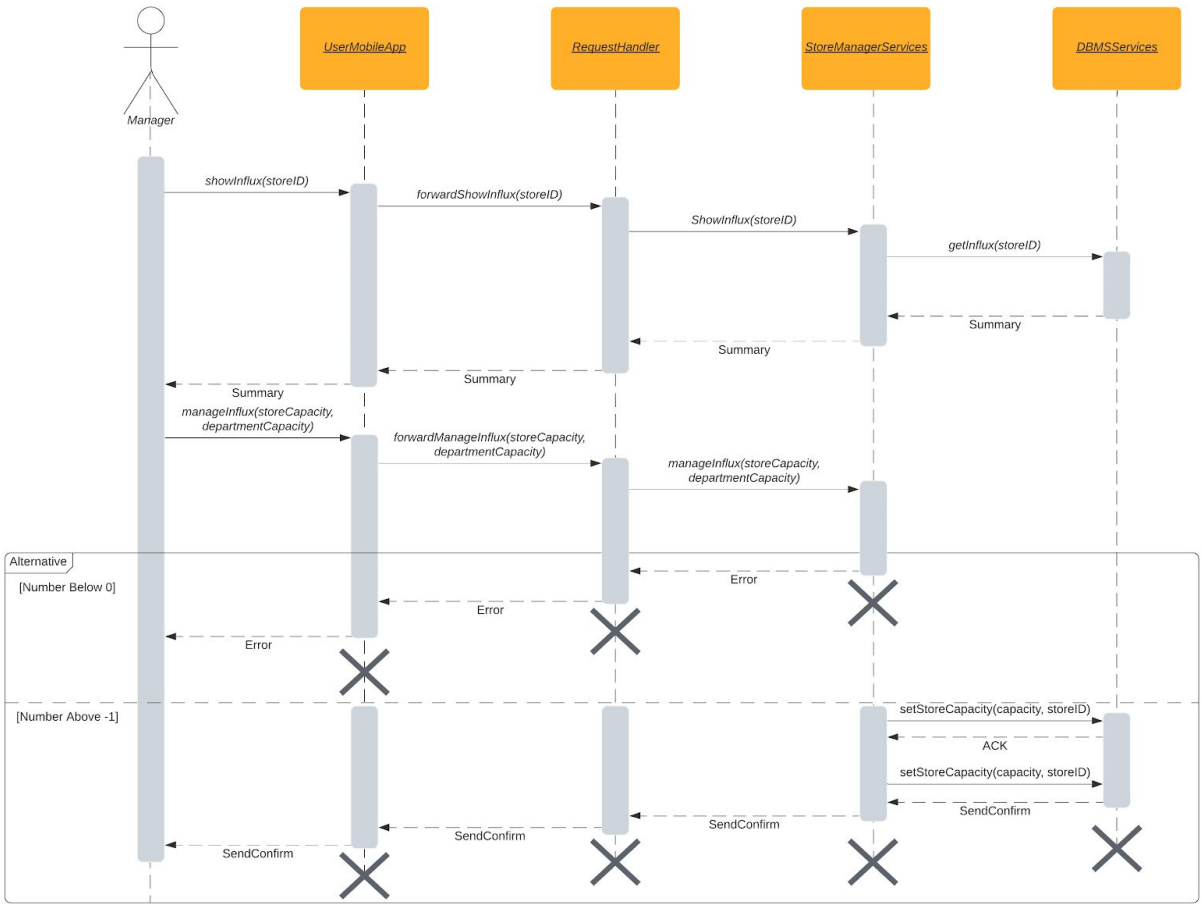


Figure 8: Check Influx View

2.5 Component Interfaces

The diagram below describes the methods that can be invoked from each interface shown in the Component Diagram.
Signatures of the methods shown below may be adapted when a future implementation is developed and further aspects will be revealed during the development process.

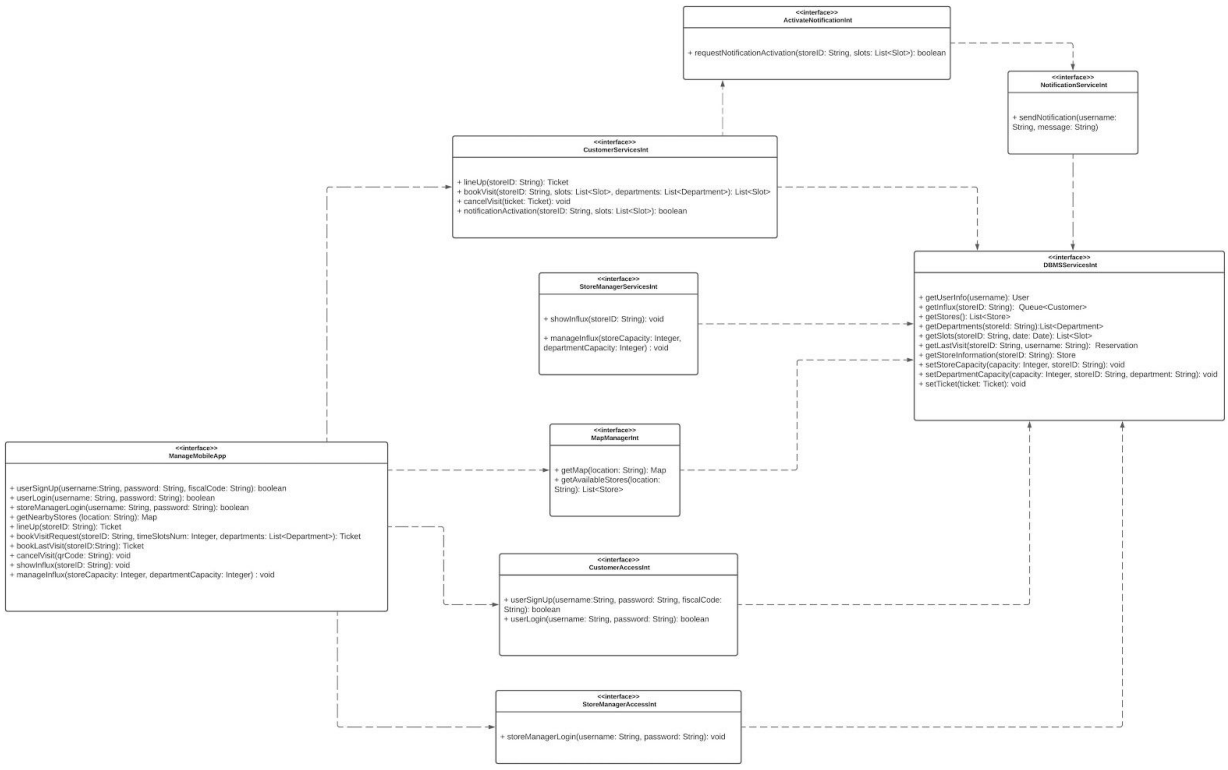


Figure 9: Component Interfaces Diagram

2.6 Selected Architectural Styles and Patterns

In this paragraph the choices of architectural styles and patterns are going to be analyzed, describing why those choices were made and how.

2.6.1 Client-Server

The CLup system needs to interact with multiple Customers at the same time and manages all their requests.
The solution chosen to tackle this problem was the client-server approach, handling the requests sent by multiple clients with a centralized server.

2.6.2 Four-Tier Architecture

An optimal way found to integrate a client-server approach in the system was to implement a four-tier architecture.
The mobile app represents the first tier, acting as the client, and communicating with the request handler, the second tier, which plays a fundamental role in bridging the client and the server.
The server, representing the third tier, manages the services used to accomplish every required feature.
It gets the data needed from the DBMS, the fourth tier, that collects and stores any useful piece of information.
This approach increases the decoupling of the system in addition to more scalability, reusability and scalability.

2.6.3 RESTful interface

In order to enforce the advantages of multi-tier architecture, we use a server-side RESTful interface that has a central role for the decoupling, scalability, flexibility and reusability of the system and its components.
In fact this type of client-server architecture allows a uniform interface for different types of clients thanks to the identification and manipulation of resources (and services) through a uniform representation of them and self-descriptive messages.
The communication is based on the HTTP protocol, taking advantage of its stateless and cacheable characteristics to implement a REST interface.

Due to the stateless property of RESTful architecture we have also to decide how to perform login/logout operations and this aspect fits perfectly with the Firebase's rest API, a token-based authentication service, described in more detail in the next paragraph.

2.7 Other Design Decisions

2.7.1 Firebase Manager

This component is designed to manage and to take advantage from the powerful services offered by Google Firebase cross-platform, in particular:

- **Authentication REST API:** a token-based authentication service that through a REST API and an HTTP POST method, permits to perform various operations like the creation of a new user, the login of the already registered users, and their modification or removal.
  It also allows to create a new user from an email and a password and login with them, to better manage Customer's authentication instead of a token-based auth that fits more for StoreManager's authentication.

- **Firebase Cloud Messaging API:** a cross-platform messaging solution that lets you send messages reliably at no cost. It permits to notify a client if some data is available to sync. In an instant messaging use case as the CLup's notification service, FCM messages is a good solution thanks to the possibility of transferring a payload up to 4KB to a client app.

3. User Interface Design

In this section are shown some mockups of the most relevant app's interfaces, first from the perspective of the Customer and then from the perspective of the Store Manager.
Below there is the login page mockup, already shown in the RASD.

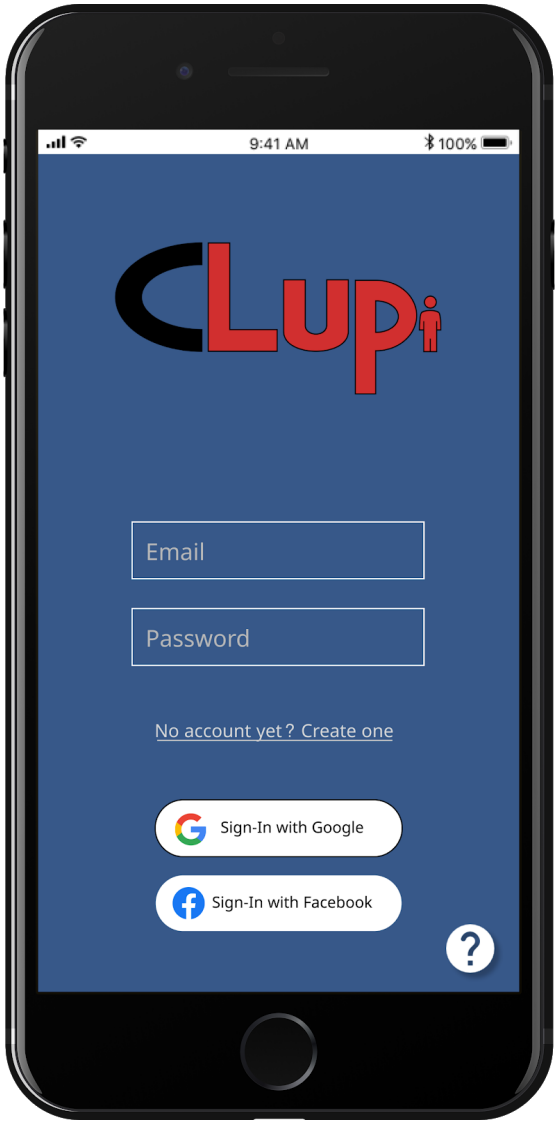


Figure 10: Login Page Mockup.

3.1 Customer Interface Design

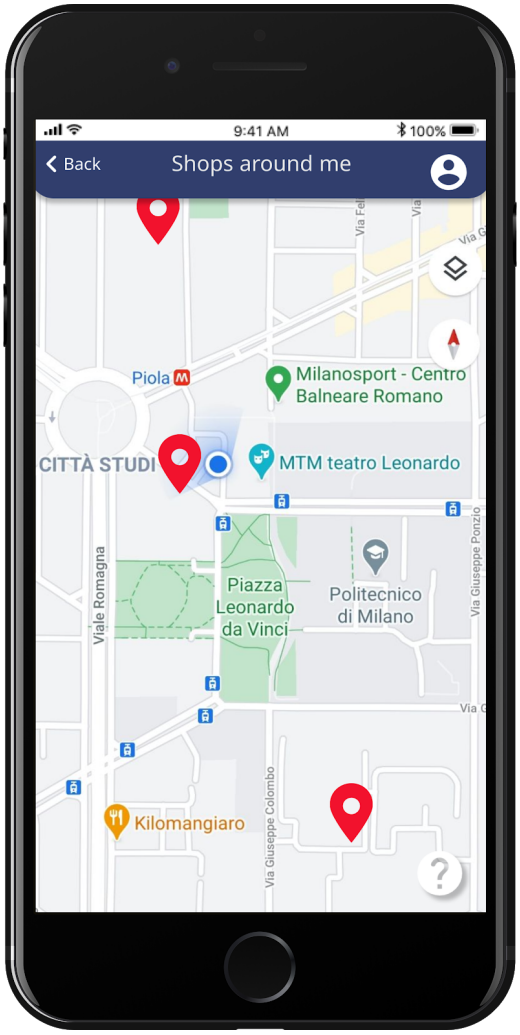


Figure 11: Map mockup.

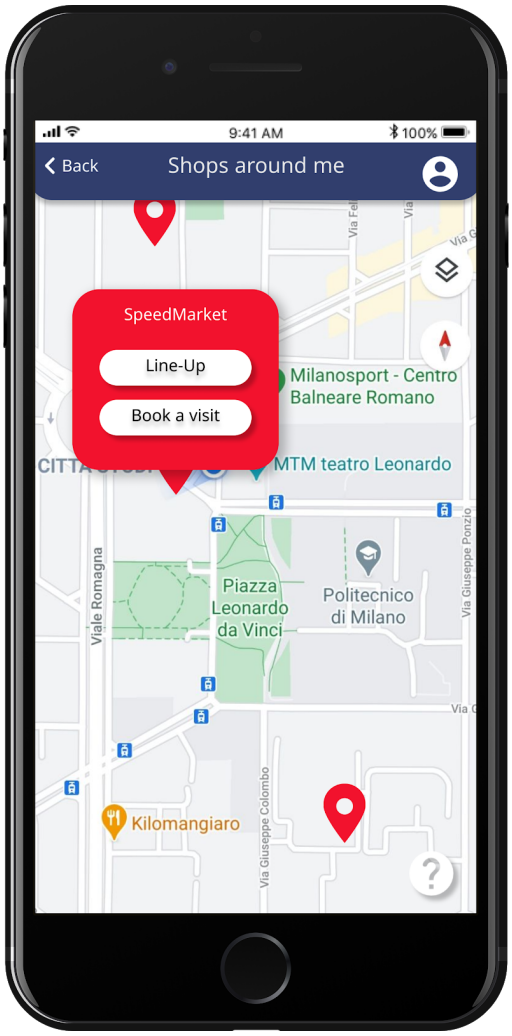


Figure 12: Store selection mockup.

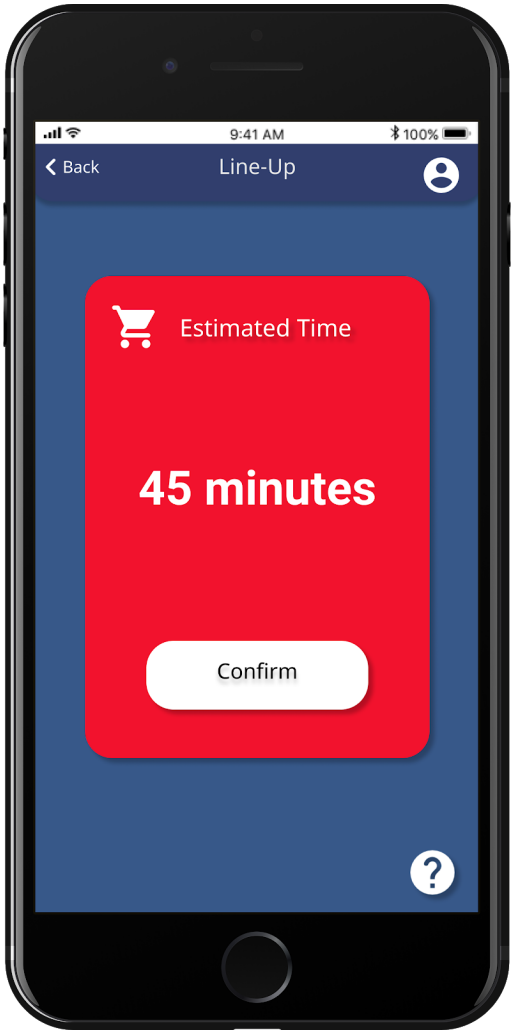


Figure 13: LineUp Mockup.

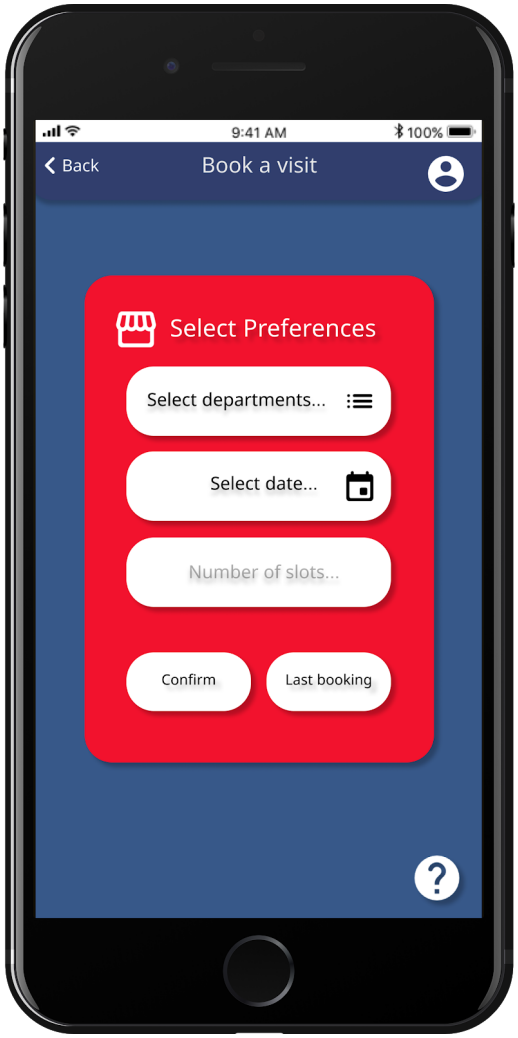


Figure 14: BookVisit mockup 1.

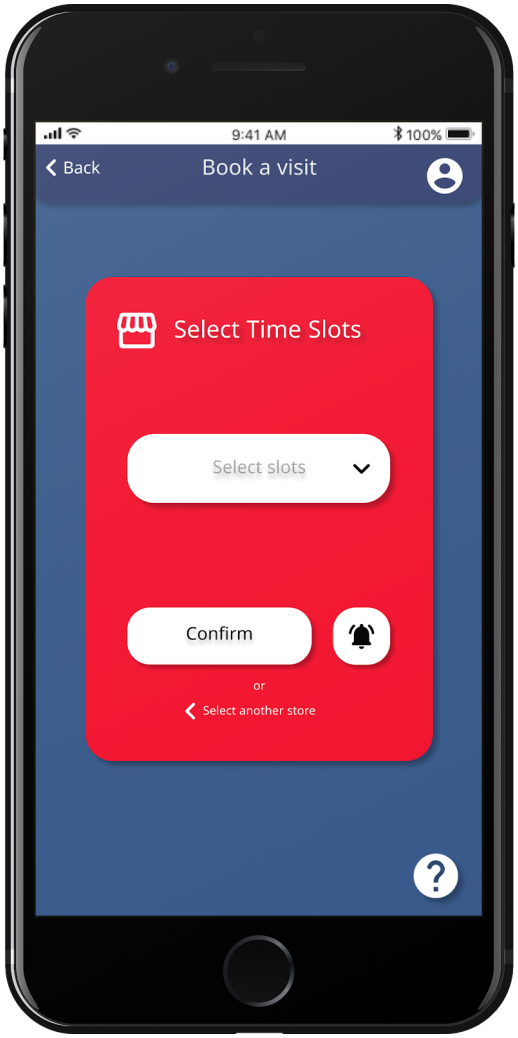


Figure 15: BookVisit mockup 2.

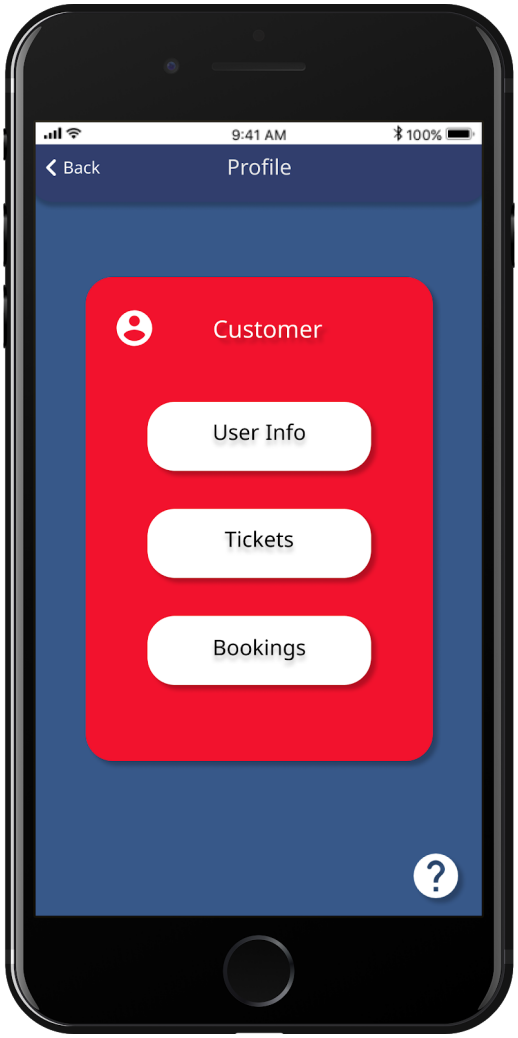


Figure 16: CustomerProfile mockup.

3.2 Store Manager Interface Design

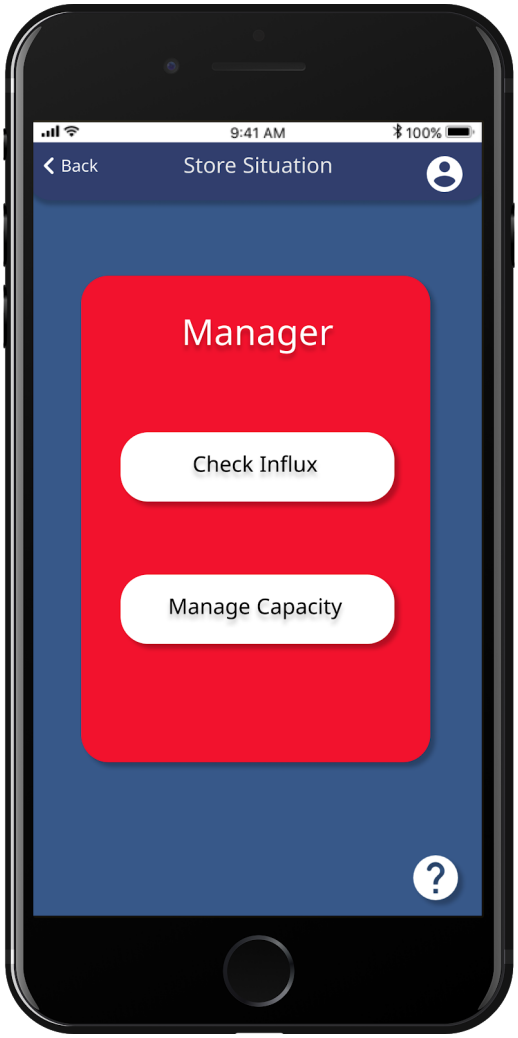


Figure 17: StoreManager functionalities mockup.

3.3 User Interface Design Overview

Below there's an overview of the interactions between the most relevant user interface's mockups:

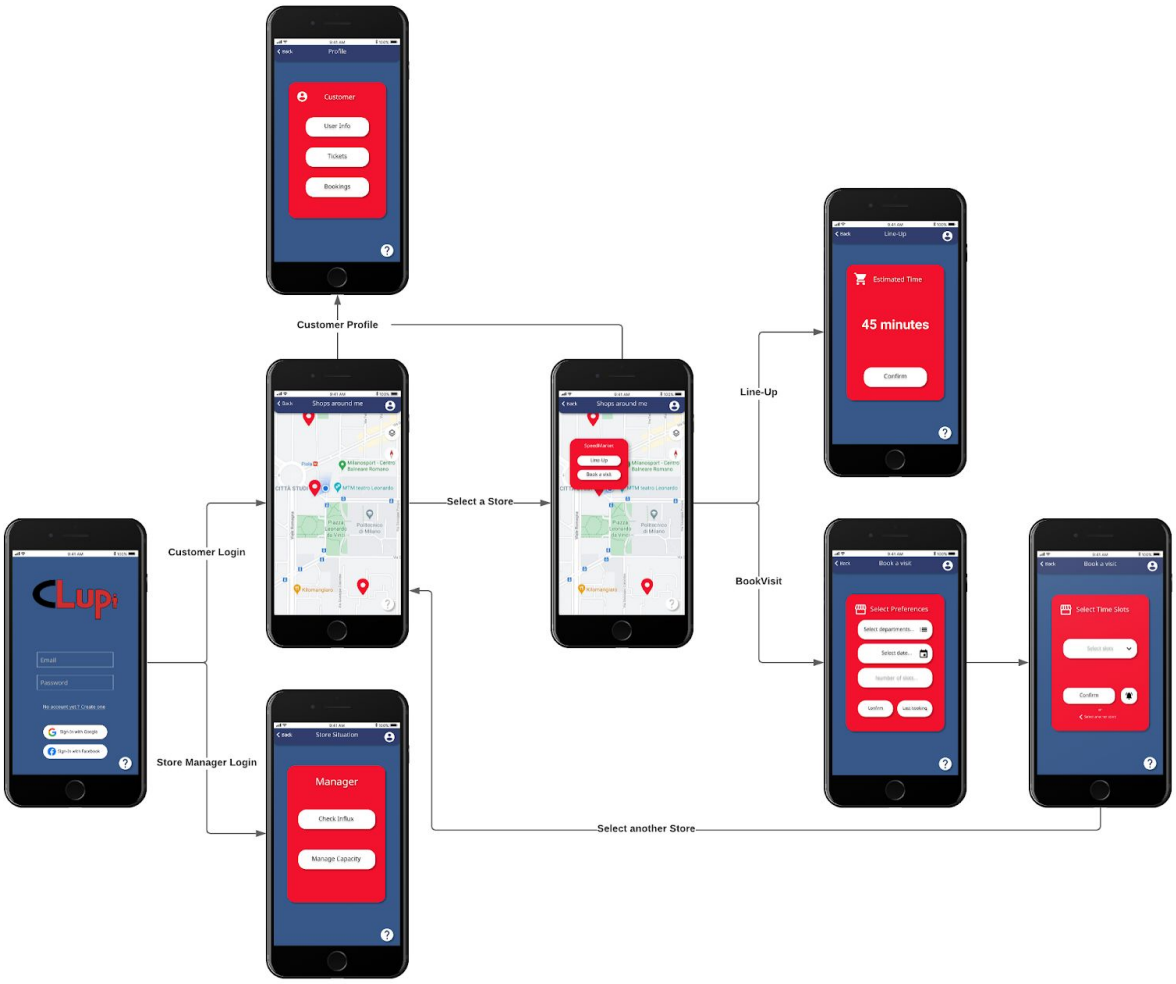


Figure 18: User Interfaces flow

4. Requirements Traceability

Below are listed the goals and for each one the requirements and components involved are shown:

G1	The application must allow a Customer to become a registered User after providing credentials and his address
	Requirements: **R1**
	Components: ● UserMobileApp ● FirebaseManager ● DBMSServices
G2	The application must allow a user to digitally line up in a queue from his home
	Requirements: **R2, R4, R5, R6, R8, R15**
	Components: ● UserMobileApp ● MapManager ● CustomerServices ● DBMSServices
G2.1	The application must allow a user to see his/her position in the queue
	Requirements: **R2, R4, R5, R6, R8**
	Components: ● UserMobileApp ● CustomerServices ● DBMSServices
G3	The application must allow a registered user to book a visit to a selected store
	Requirements: **R2, R4, R5, R7, R9, R10, R11, R12, R13, R17**
	Components: ● UserMobileApp ● MapManager ● CustomerServices ● DBMSServices

G3.1	The application must allow a registered user to select departments in which he wants to buy
	Requirements: **R2, R9, R10**
	Components: ● UserMobileApp ● CustomerServices ● DBMSServices
G3.2	The application must allow a registered user to choose a proposed time slot for the visit
	Requirements: **R2, R11, R17**
	Components: ● UserMobileApp ● CustomerServices ● DBMSServices
G4	The application must allow a user to retrieve a QR code in case of “virtual lineup” or “book a visit”
	Requirements: **R2, R15**
	Components: ● UserMobileApp ● CustomerServices ● DBMSServices
G5	The application must allow a Customer to physically line-up in front of the store
	Requirements: **R14**
	Components: ● StoreClient ● CustomerServices ● DBMSServices

G6	The application must allow a Store Manager to manage the number of customers for each supermarket department
	Requirements: **R3, R9, R17**
	Components: ● UserMobileApp ● StoreManagerServices ● DBMSServices
G7	The Application must allow the Store Manager to monitor entrances
	Requirements: **R3, R22**
	Components: ● UserMobileApp ● StoreManagerServices ● DBMSServices
G8	The application should allow a registered Customer to cancel a visit previously booked
	Requirements: **R19, R20**
	Components: ● UserMobileApp ● CustomerServices ● DBMSServices
G9	The application should send the Customer notifications when slots are available, if requested
	Requirements: **R18, R21**
	Components: ● UserMobileApp ● NotificationManager ● FirebaseManager ● DBMSServices

Table 1: Requirements Traceability table

- **R1**: A Customer must be able to begin the registration process. During the process the system will ask the Customer to provide credentials.
- **R2**: A Customer must be logged in to the system using their credentials.
- **R3**: Store Manager must be logged in to the system using their credentials.
- **R4**: The system must be able to provide the list of stores on the map around the Customer
- **R5**: The system must be able to let the Customer select a store from the ones available on the map
- **R6**: A Customer must be able to choose to lineup for a store
- **R7**: A Customer must be able to choose to book a visit at a store
- **R8**: A Customer must be able to check the current position in the line to enter the store that he chose.
- **R9**: For each store, the system must be connected with a DB listing all the departments in that store.
- **R10**: During a “book a visit” process, the system must be able to suggest departments to the Customer based on previous visits to the store.
- **R11**: A Customer must be able to select a time slot from the available ones.
- **R12**: The system notifies the Customer 60 minutes before the booked visit to the store.
- **R13**: If the Customer chose “book a visit”, the system must be able to show the estimated time it takes to the Customer to reach the store.
- **R14**: The system must be able to connect to a printer and to print tickets.
- **R15**: The system must be able to show the ticket number and QR Code if a virtual ticket is taken.
- **R16**: Based on the number of people inside the store, an authorized store Manager must be able to regulate the influx of people coming in the store
- **R17**: The system must communicate with a DB that contains available time slots to “book a visit”
- **R18**: A Customer must be able to select a time slot from the non-available ones.
- **R19**: A Customer must be able to view previously booked visits.
- **R20**: A Customer must be able to cancel a previously booked visit.
- **R21**: A Customer must be able to receive notifications for available slots in the future.
- **R22**: A Store Manager must be able to monitor entrances through the “check influx” feature.

5. Implementation, Integration and Test Plan

This section will lay out the plans for Implementation, Integration and Testing of this project in order to cover all the features that our system needs to accomplish.
In the first paragraph our implementation and testing approach will be introduced. Later with the use of the **utility tree** we will show the importance of each feature related with the difficulty of its implementation.
Finally an overview on the different types of testing that our system will undergo will be provided.

5.1 Strategy Overview

This project will be implemented adopting a **bottom-up** approach. This is because without implementing first critical components such as **DBMSServices** and most critical server-side components it will be very difficult to test and implement all the components that come after that.
This approach will be used for both the client side and server side.
After the implementation and integration test of the core functionalities (line-up and book a visit services) we can proceed to test the interactions of them with the external services.
About these last ones, it's irrelevant to test the methods that they provide because as external services we can assume that they are already tested.
The entire plan is based in a parallel implementation and testing approach, to identify as soon as possible errors and bugs in the system and to reduce the costs for their correction.
In order to support the bottom-up approach we need to use drivers to support the incremental phases of test planning.

Development tools: considering the application nature and the number of systems it needs to run on the programming language Java is considered to be the best option thanks to the large amount of libraries and frameworks to support the developing with it on different platforms.

Testing tools: powerful testing tools are available for OO programming, in our case for Java, we are going to use some of them as support for the testing plan:

- **JUnit:** a framework for unit testing to write repeatable tests in Java.
  It also allows to work with other testing tools and it is integrated with all major IDEs.

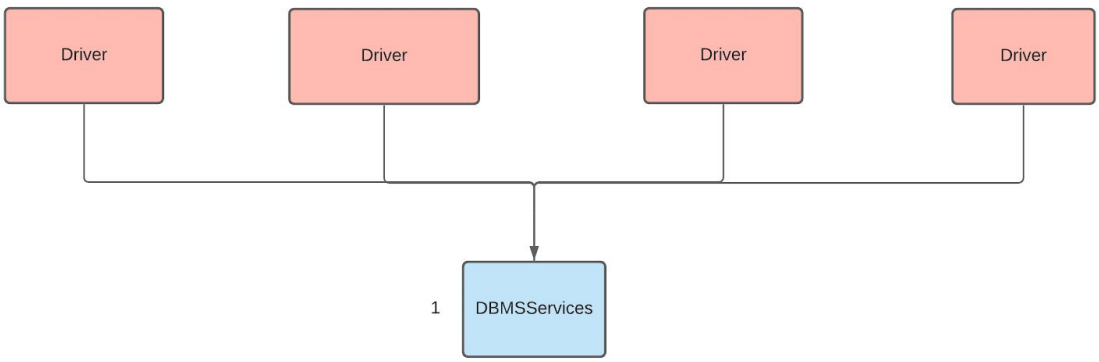
- **Mockito:** a mocking framework for Java classes to test code in isolation without dependencies.
  It is perfectly integrated with JUnit.

- **REST assured:** a tool for REST API integration test that we will use to test and validate the implementation of our RESTful interface with the support of an HTTP CLient.

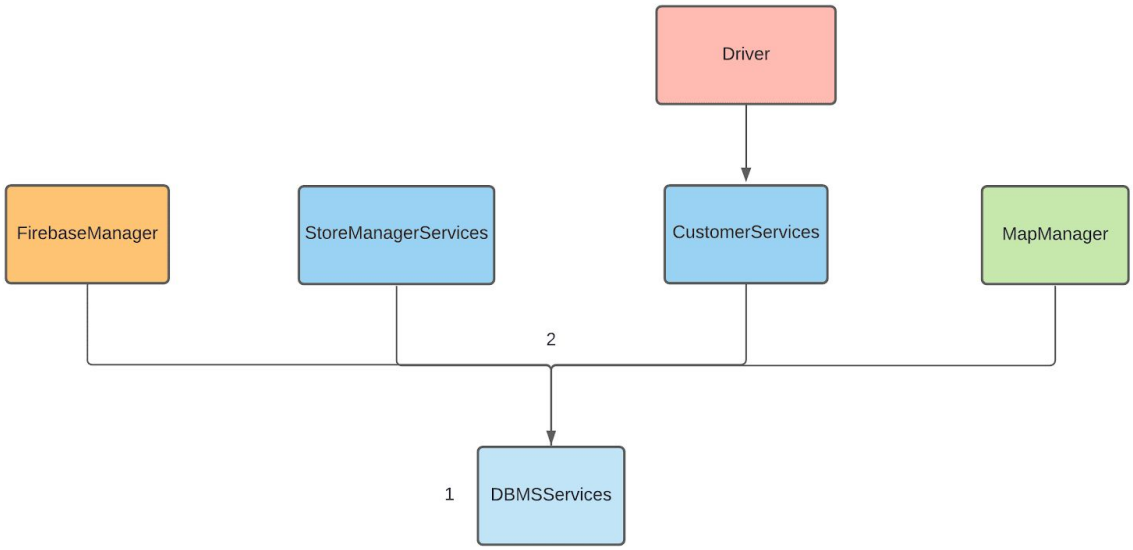
5.2 Implementation Plan

Discussed here is the order in which all the components will be implemented and tested.
As said above, the approach chosen for the implementation is the bottom-up one, so the components will be added gradually, from the base up to the top ones.

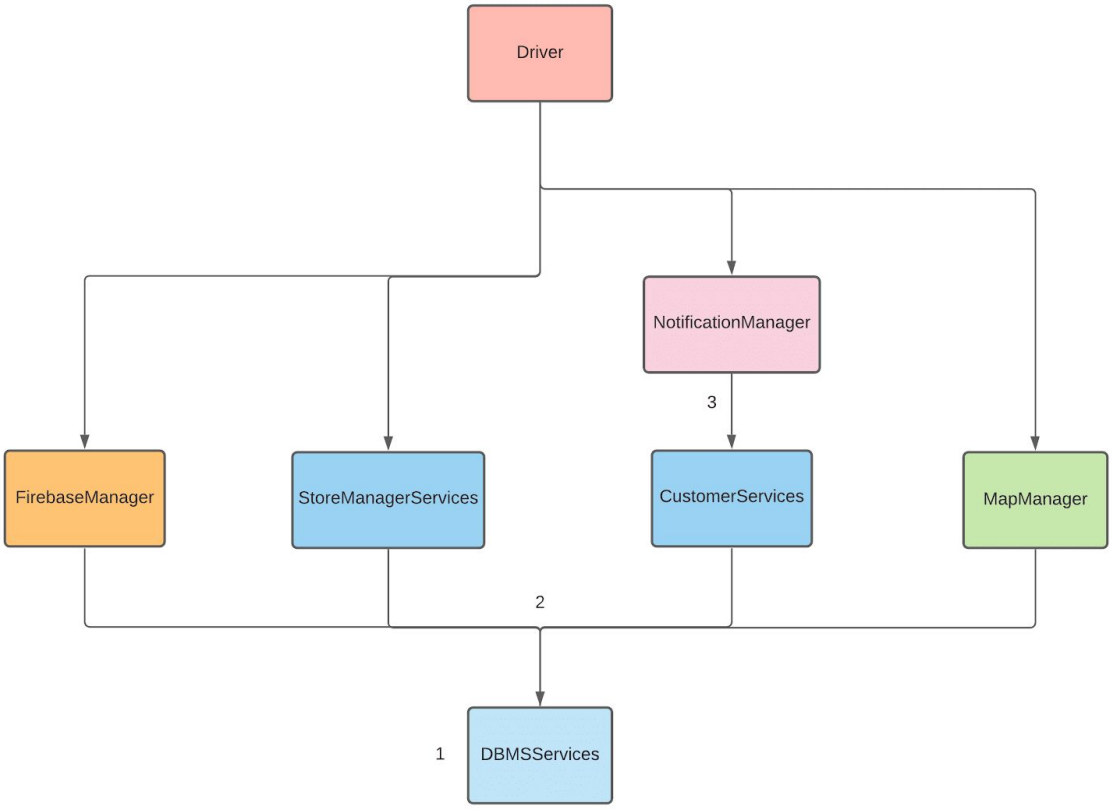
1. The first component added will be the DBMS, as it will integrate all the data management functions that will be necessary for all the other modules.



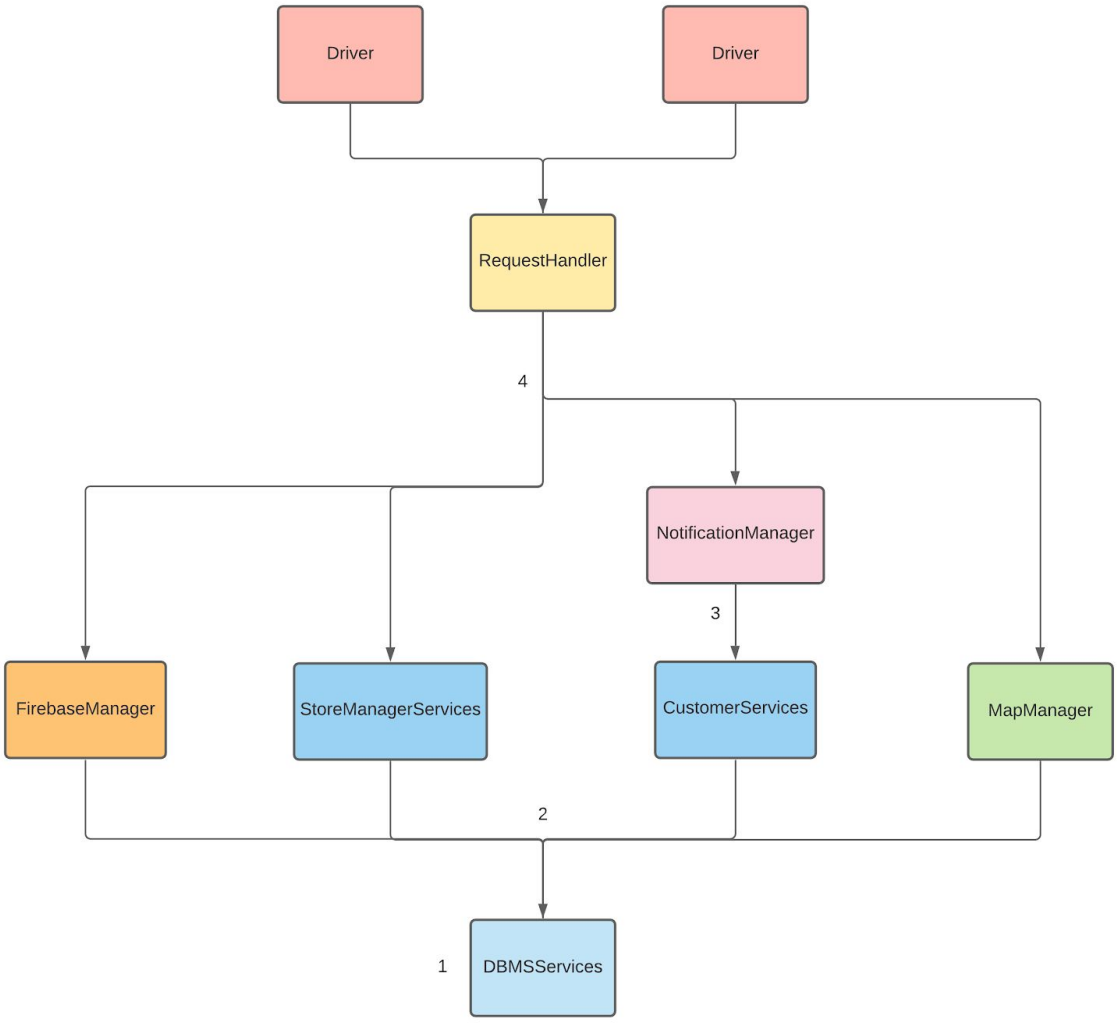
2. Then the CustomerServices and the StoreManagerServices will be added, as they integrate the core functions of the application, like lining-up or managing the entrances to the store.
   Specifically the CustomerServices will be needed in the next step to integrate the NotificationManager.
   Then MapService and FirebaseManager with its login, register and notification function are integrated.
   All these modules can be added independently as they don't interact with each other.
   Regarding the MapService and FirebaseManager, since they are external services, only the integration test will be done because all the functions bound to these components are already tested.



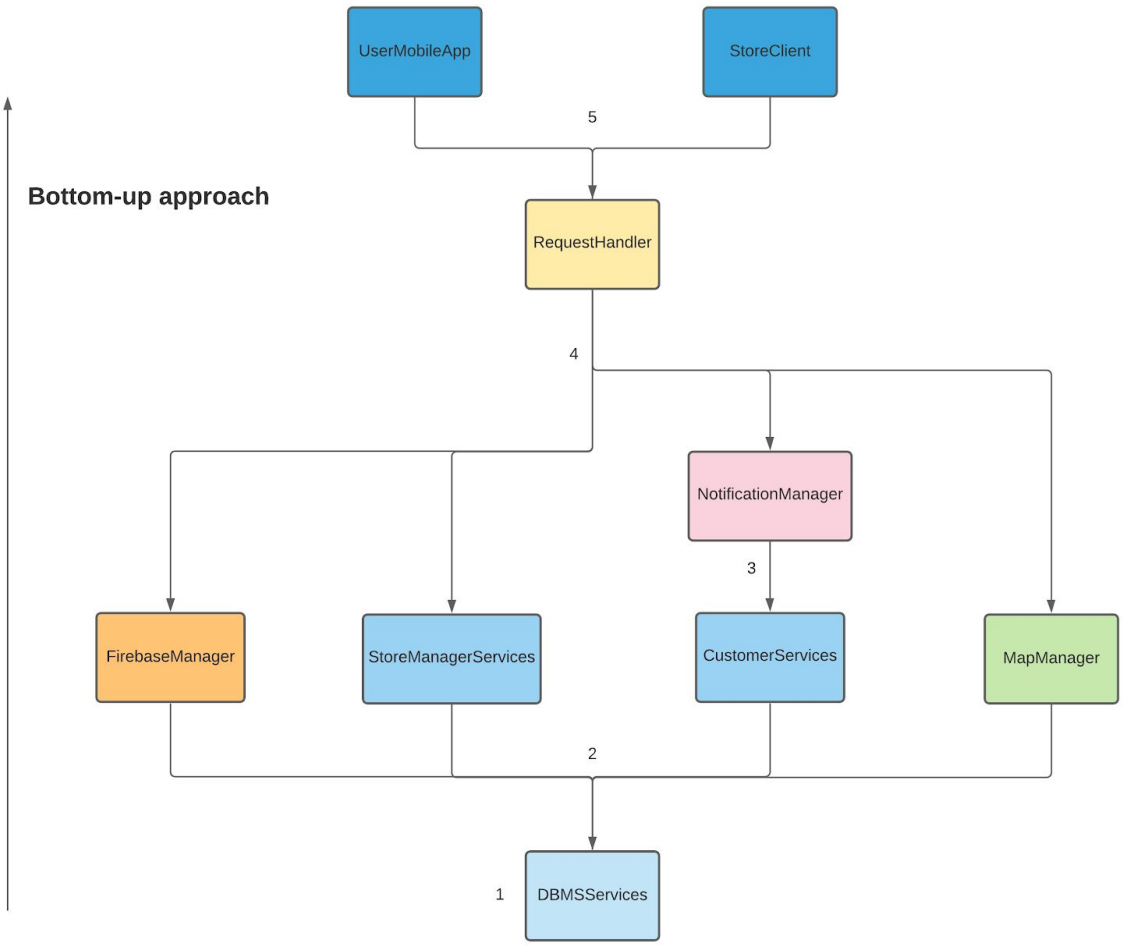
3. At this step, regarding the notification function of the “book a visit” feature, the NotificationManager needs to be added after the CustomerServices.



4. The RequestHandler is added after these modules as its goal is to sort messages to the respective module, all already implemented.



5. Finally, the UserMobileApp and the StoreClient are added, with their function and interaction with the RequestHandler.



5.3 Utility Tree

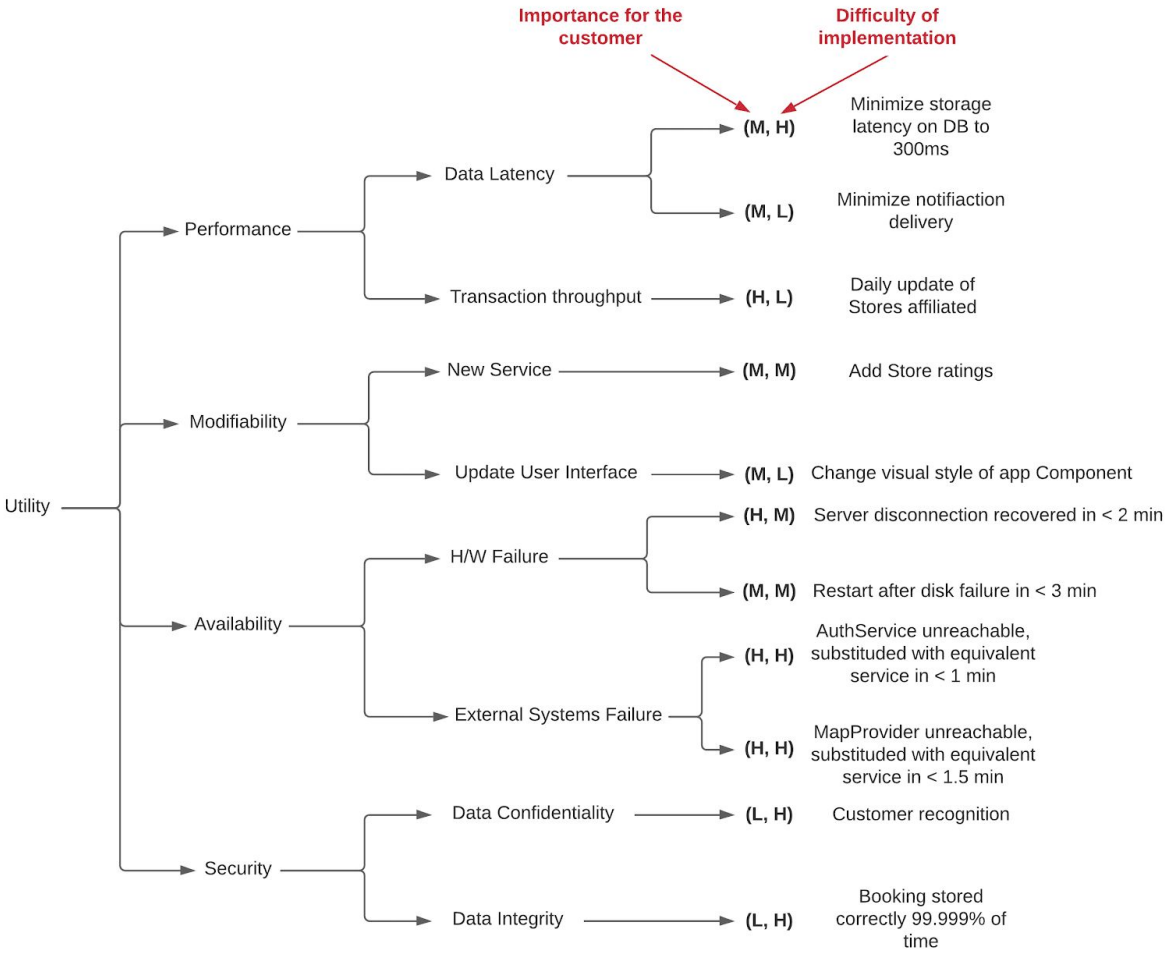


Figure 19: Utility Tree

5.4 System Testing

When all components and their functions have been tested, the system as a whole needs to be analyzed in various aspects in order to be sure the application works as intended.
This can be achieved through some test types, as described below:

- **Functional testing:** It is necessary to be sure that every requirement is satisfied during the process
- **Performance testing:** Useful to understand where the system is not efficient, checking where the application slows down during all kinds of activities
- **Load testing:** The system is asked to handle multiple requests in order to check if it manages all of them as planned, focusing particularly on memory leaks or misuse of it
- **Stress testing:** This test overloads the system with work, amining to determine the stability and acknowledge its breaking point

6. Effort spent

Topic	Hours
Introductory discussion on DD	3h
Architectural design overview and high level components	6h
Component Diagram	5h
Component Interfaces	3h
Design pattern and Other design decisions	4h
User Interface Design Mockups	12h
Implementation and Integration Test Plan	6h
Revision of DD	2h

Table 2: Effort spent by Giampiero Repole

Topic	Hours
Introductory discussion on DD	3h
Component Diagram	9h
Deployment View	7h
Component Interfaces	4h
Requirements Traceability	6h
Implementation and Integration Test Plan	8h
Revision of DD	2h

Table 3: Effort spent by Andrea Zanetti

Topic	Hours
Introductory discussion on DD	3h
Component Diagram	2h
Runtime View	12h
Design pattern	3h
Implementation and Integration Test Plan	5h
Revision of DD	2h

Table 4: Effort spent by Pasquale Occhinegro

7. References

- All diagrams have been made with https://lucid.app/
- Mockups have been made with https://www.figma.com/