POLITECNICO
MILANO 1863

DD
Design Document

AY 2020-2021

Students:

Matteo Makovec
Lorenzo Male
Gabriele Morelli

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give more technical details than the RASD about the CLup application system. While the RASD presented a general view of the system and what functions the system is supposed to execute, this document aims to present the implementation of the system including components, run-time processes, deployment and algorithm design. It also presents in more details the implementation and integration plan, as well as the testing plan.

In particular, the following topics are touched by the document:

- The high-level architecture
- The main components, their interfaces and deployment
- The runtime behavior
- The design patterns
- The user interface
- A mapping of the requirements on the architecture's components
- Implementation, integration and testing plan

1.2 Scope

In order to face the problem of social distancing, the software wants to give the possibility to users to queue and book visits for stores from home, avoiding crowds outside of them; at the same time, the software regulates the influx of people inside the building, avoiding crowds inside stores, giving to store managers the possibility to monitor entrances. Finally, the software can speed up the process, always guaranteeing safety distance between people.

1.3 Definitions, acronyms, abbreviations

DEFINITIONS:

TERM	DEFINITION
Employee	He/she works in a store and he/she does the functionality of “mediator”, performs the “physical booking” functionality for customers and scans the QR codes at the entrance and at the exit
Store Manager	He/she is in charge of monitoring his/her store and subscribes the store to the CLup system sending to the CLup team the proper documentation
Booking	It allows the user to book a visit for a specific store
Ticket	It allows the user to queue for a specific store
Physical booking	It allows the user to book physically a visit for a specific store
Visit	It represents the “user journey” internally of the store, tracking his/her entrance and the exit
EstimatedDuration	It is the user’s estimation of his/her visit duration
Store	It is the supermarket subscribed to the CLup system
Device	It is a hardware component used by the actors to use our software, it can be a smartphone or a tablet
CLup application	It defines the mobile application downloaded in the user’s device
Customer	It is the client of a store who uses the “physical booking” functionality
User	It is the client of a store that uses the CLup application
Estimated waiting time	It is a estimation of the time that the user has to wait for his/her turn
Recommended departure time	It suggests the user the best time to begin moving from the starting address in order to arrive right on time at the store
Starting address	It is the address from which the user will begin moving to go to the store
Maximum capacity	It is the maximum number of people admitted at the same time in a store
Queue	Virtual sequence of users waiting for their turn
Tuple	The concrete entity saved in the Data Base, it’s a sequence of fields

ACRONYMS:

TERM	DEFINITION
GPS	Global Positioning System
RASD	Requirement Analysis and Specification Document
QR code	Quick Response code
DD	Design Document

SYNONYMS:

TERM	SYNONYM
Mobile app	Synonym of CLup application

1.4 Revision history

DATE	MODIFICATION
10/01/2021	Release
12/01/2021	New release adds the traceability of the requirement added in RASD2

Changes performed in DD2:
Changes have been made in the traceability section (section 4) to follow changes made in the RASD2.

2 Architectural Design

2.1 Overview

The CLup system we're going to develop provides many functionalities, here it's worth to just mention some of them for sake of understanding of the architectural choices made. First, to take a ticket. There are many crucial aspects of this, one of them is the fact that the CLup application provides to the users the estimated waiting time for his/her turn to come. This is calculated by the system using different parameters, some of them provided by the user. Second, to do a booking in order to reserve your visit. It's worth to underlie that If the user tries to book a slot of time which is already busy, besides the fact he/she won't be able to book it, the system gives back a suggestion of possible

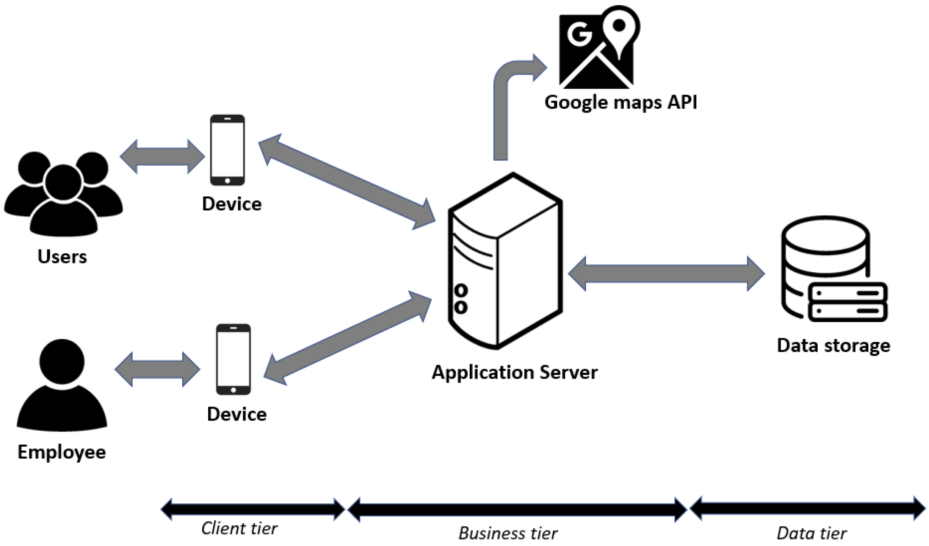
alternatives, so a control performed by the system is needed. Besides, the system allows stores to book a visit for customers that go physically to the spot, it's clear that many actions performed by the system are needed to provide this feature. All these three features use QR codes to uniquely identify users inside the CLup system and to aid the work of the store manager in regulating and managing the influx of people. This last action is actually performed by the store employees checking the validity of a memo by crossing the scanned QR code with the information retrieved from the data storage. Since the architecture of a software system defines the system in terms of computational components and interactions among them, the architecture has to be made client-server style, since we have a side which mainly uses functionalities provided by the other side. Then, we've decided to separate the whole system in further logical parts, to further decouple it, obtaining a concrete division of the roles and the duty of each part, this has been done also to achieve an high degree of maintainability and portability, ending-up to go for a three-tier architecture as will be described more detailed in the following section.

2.1.1 High level components

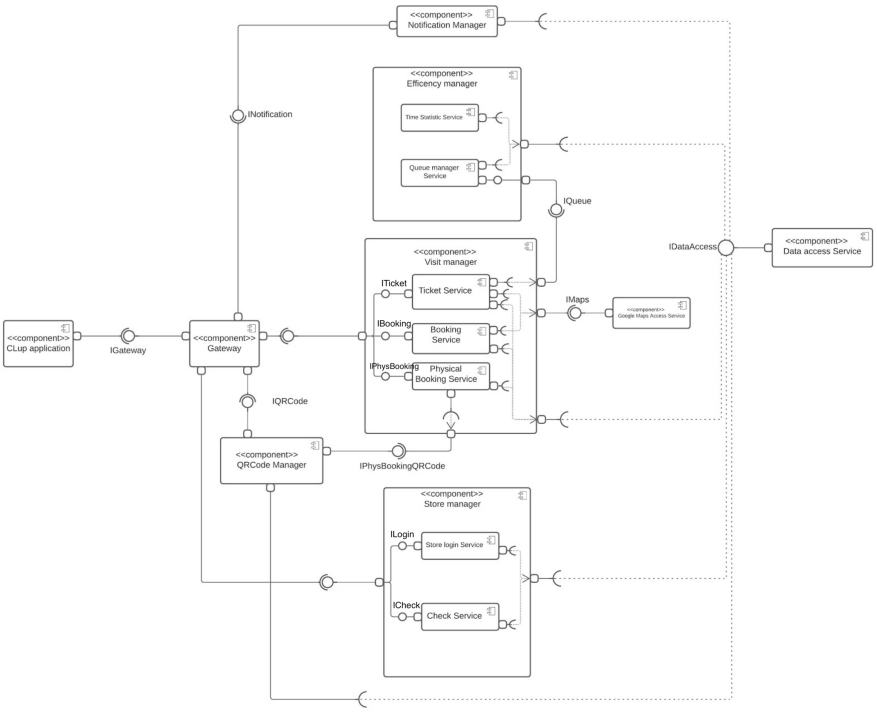
The architecture of the CLup application is structured according to the 3-Tier architecture:

- **Client tier:**
  handles the interaction with the user providing him/her the interfaces to interact with the business tier and provides the content displayed on the user interface
- **Business tier:** it's in charge of the logic of the entire system, concretely providing the functionalities and includes methods to interact with the data storage
- **Data tier:** it's responsible for the persistency of the information, it actually has the access to the data storage.

Separating logically the tiers from one another allows locating the different parts onto different machines, that's why this approach will be used also to physically split the code mapping it, depending on the function performed, onto three main hardware components. The three main hardware components are: the device on which is installed the mobile app, the application server and the data storage. The first, which here acts as the "client tier", it's used by the user, or the employee, to interact with the business tier. The second, which here acts the "business tier", it's in charge of the entire logic, performing operations and communicating with the data storage. The third, which here acts as the "data tier", persists data.



2.2 Component view



Since we've adopted a three-tier architecture for our CLup system, the following component diagram gives a view of the system focusing on the representation of the internal structure of the application server, highlighting how its components interact.

- **CLup application:** it isn't part of the application server, but it was worth to represent it in order to highlight which is the component that allows the user or the employee to interact with the application server. Besides, it's notable it's dependency with the QRcode Manager which allows it to obtain a new unique QR code every time a new mobile app is downloaded

- **QRCode Manager:** this component generates and associates the QR codes when a new mobile app is downloaded or when a customer does a physical booking at the store

- **Notification Manager:** this component is meant to notify long-term customers suggesting them on when to book a visit. This component periodically retrieves all the information needed, about a specific user, to create a suggestion, which will be provided to him / her

- **Efficiency Manager:** his component consists of two sub-components which provide the functionalities to make the system work efficiently. First, the "Time Statistic Service", which is in charge of doing statistics on the time that a long-term user declared that his / her visit would have lasted when he / she booked it and the time actually spent during such visit. This component retrieves and elaborate these information to estimate how accurate is the estimated duration such long-term user stated. Second, the "Queue Manager Service", this component has multiple functionalities: it calculates the estimated waiting time for a specific store, enhances the store capacity when possible and performs the dynamic managing of the queue using also the time statistics

- **Visit Manager:** this component is meant to provide the most important functionalities of our software: take a ticket, book a visit and do a physical booking. The first is realized by the "ticket service" sub-component, which can interact with the "queue manager" to obtain the estimated waiting time. The second is realized by the "Booking service", which is also in charge of providing to the user the calendar with the time slots. These last two components can also interact, through the "Google Maps access service", with the Google maps API to perform all the operations that need the location. The third, is realized by the "Physical booking service", which interacts with the "QRCode Manager" to obtain the temporary QRCode and with the "Store Login Service" to ensure that is actually a store that is performing this operation.
  Besides, this component also allows the user to see the memo from a dedicated menu in the CLup application. Since the "Memo" 's aren't actually saved in the data storage, this component is in charge of retrieving all the information needed to compose it and send it back to the mobile app. Besides, it permits to cancel the

memos and so the booking or the ticket. Who concretely performs these features about the Memos are the respective subcomponents.

- **Google Maps access service:** this component is meant to connect and communicate with the Google maps API
- **Store Manager:** this component is meant to provide the two main functionality to a store, actually to the employee which physically performs them. The first subcomponent "Store login service", allows the employee to sign into the application so that he / she can scan QR codes and do physical bookings. The second subcomponent: "Check service" allows to check the validity of a memo by crossing the scanned QR code with the information retrieved from the data storage
- **Data access Service:** this component is meant to allow every other component to interact with the data storage. The interfaces provided by this component contains all the methods that can be invoked by the other components of the system to retrieve the information that they need
- **Gateway:** this component dispatches the requests and calls to methods from the CLup application to the right component of the application server. Every method is redirected to the proper component that can handle it. Besides, all the responses and data sent back pass through it to reach the mobile app.

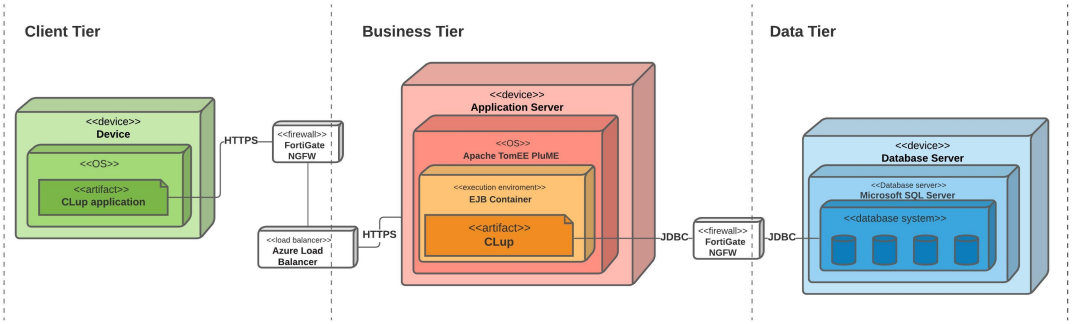
2.3 Additional specifications

The configuration adopted is of type: fat client, in which, in addition to the presentation logic, in the client node is allocated also a part of the business logic. This can be an advantage for the mobile app that is able not only to display but also to compute functions.

The gateway in the application server is a bottleneck, since it acts as the entry-point to our application server, and so a possible single-point-of failure. Furthermore, the Application server itself is a critical part of our system: it has to handle heavy workloads while maintaining a high level of performance and encapsulate the gateway component. For these reasons, a series of precautions are taken in order to increase the reliability and the performances of the system: the application server will be replicated in parallel and a load balancer is added to route the requests to the application Servers distributing the workload.

2.4 Deployment view

In the following image it can be seen the deployment diagram that shows the structure of the run-time system. The components involved are:



- **Device:** this is the application used by the user or by the employee. The mobile application is available for every Android, IOS and Microsoft device to allow compatibility with most of the devices.
  The mobile app communicates with the Application Server via JSON over HTTPS (RESTful web service).
  Using the HTTP(s) protocol, the overhead is pretty small and the connection can be secured (via SSL).
  Using a RESTful web service through the JSON format allows us to have small overhead and a well standardized environment.

- **Application Server:** this node has the business logic of the application. The application server uses EJB (Enterprise Java Beans), to ensure the functionalities of the application, and communicates with the Database Server through the TCPS protocol, to access the data.
  It has been chosen Apache TomEE PluME, that is a software platform for running web applications developed in the Java language.

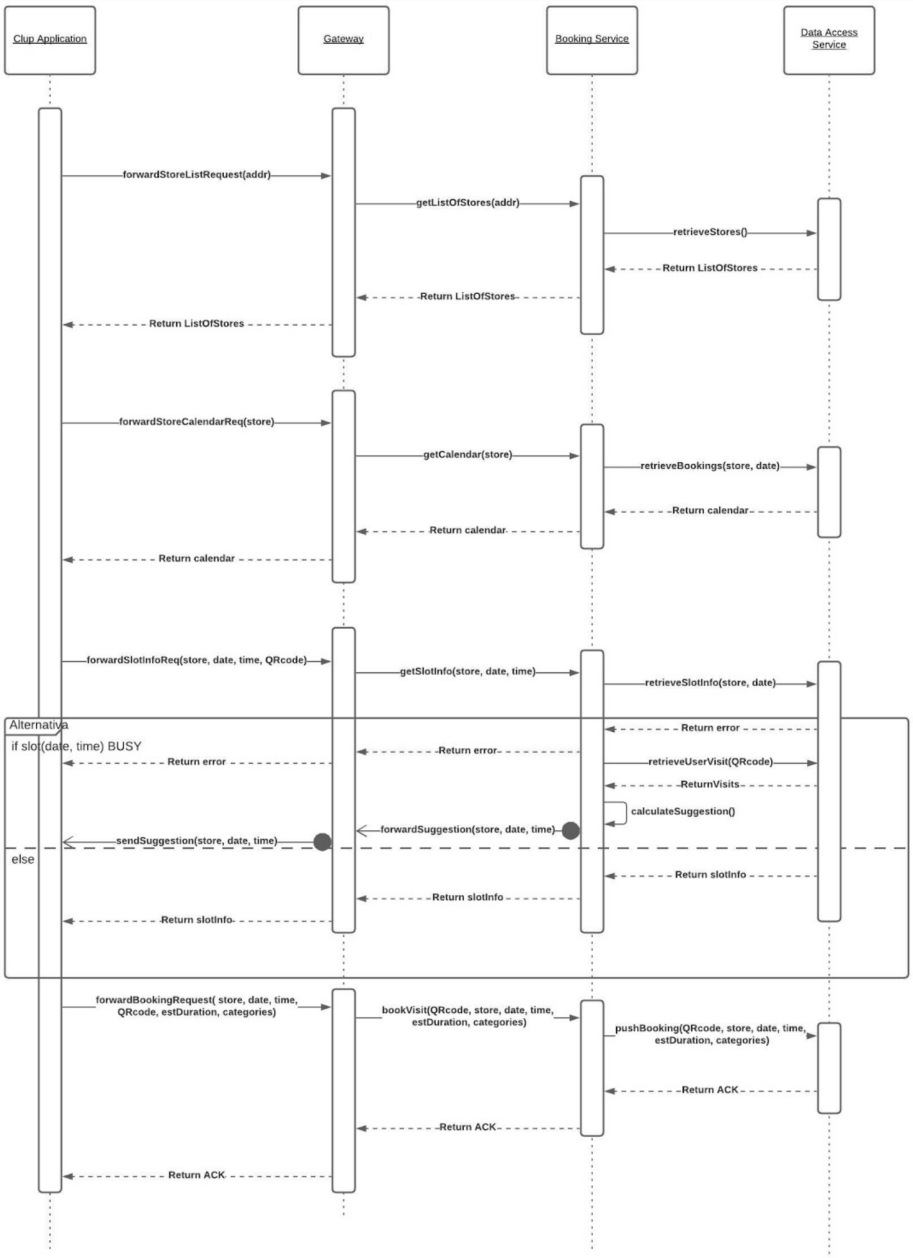
- **Database Server:** this node is the one involved in the data access. It has been chosen Microsoft SQL Server, that is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications.

- **Firewall:** since the application works with sensitive data, such as the current position of people, great attention must be paid to safety. With respect to the latter, the Firewalls have been implemented to isolate each tier and to add a level of security.
  FortiGate NGFW offers a network security platform, designed to deliver threat protection and performance with reduced complexity.

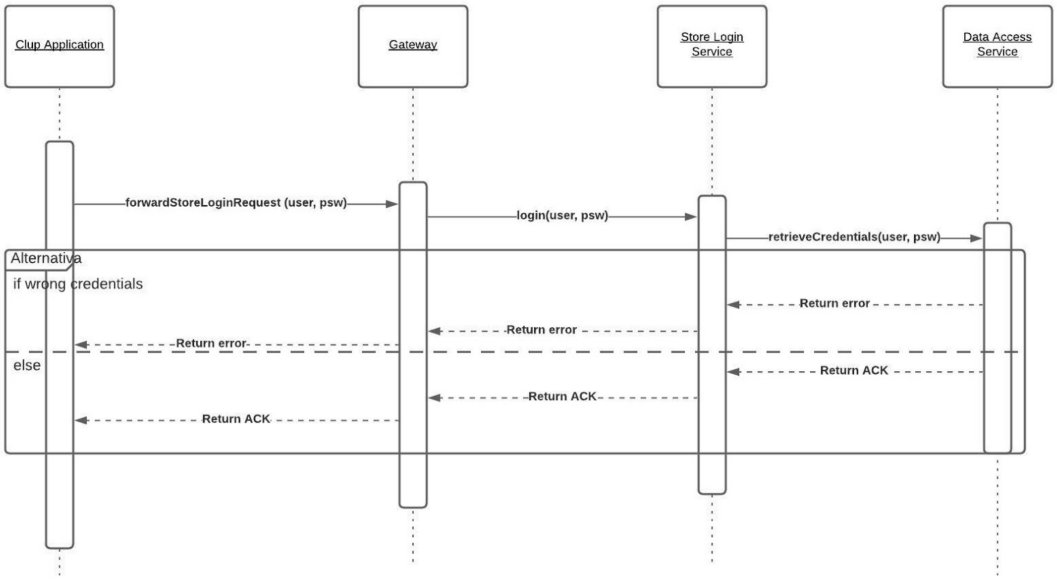
- **Load balancer:** the implementation of a load balancer is needed in order to handle lots of users/employees, avoid single point of failure, routing the request to the first Application Server available and increase the performance.
  The Microsoft Azure Load Balancer is a built-in load balancing solution for internet-facing applications. It allows users to build highly-available and scalable web applications and load balance internet and private network traffic.

2.5 Runtime view

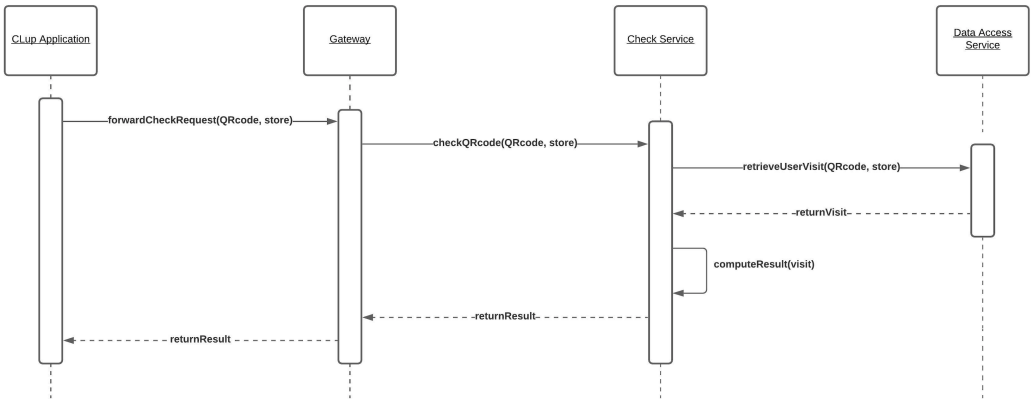
2.5.1 Book a visit



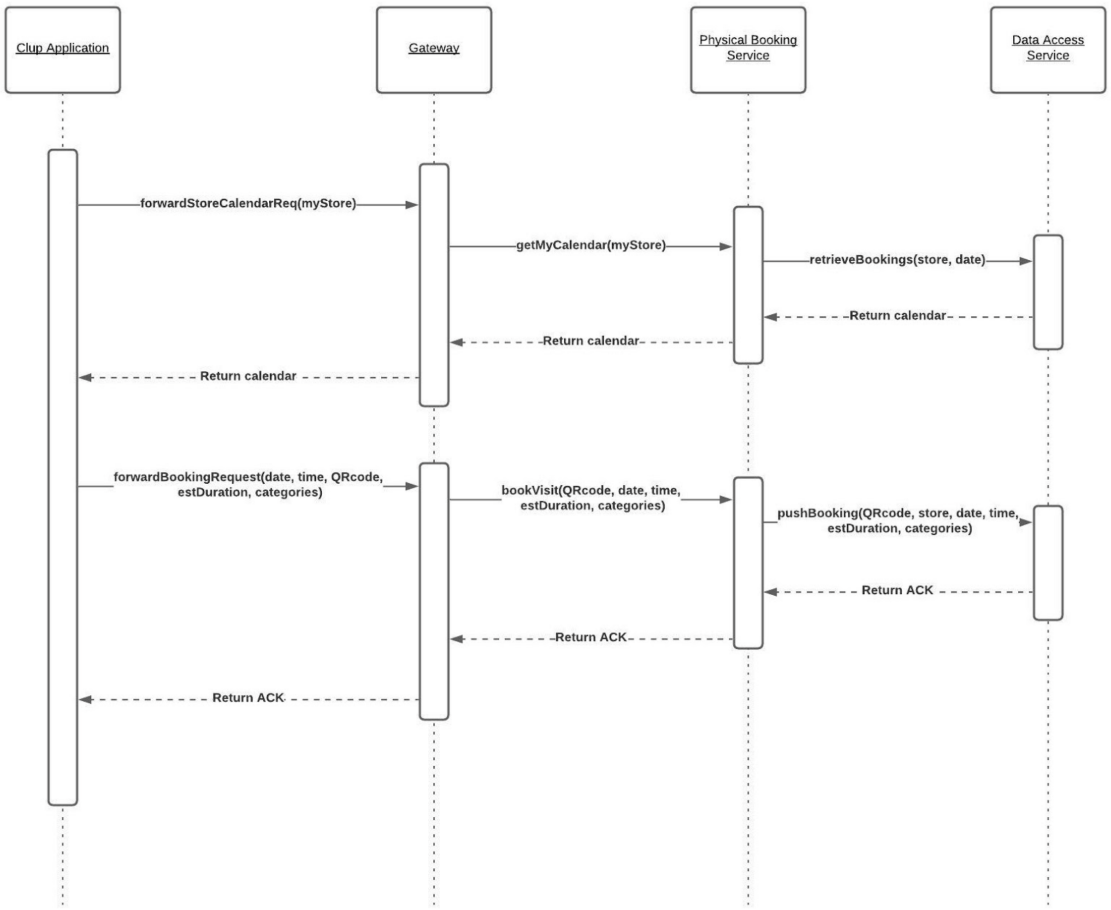
2.5.2 Sign-in the CLup application



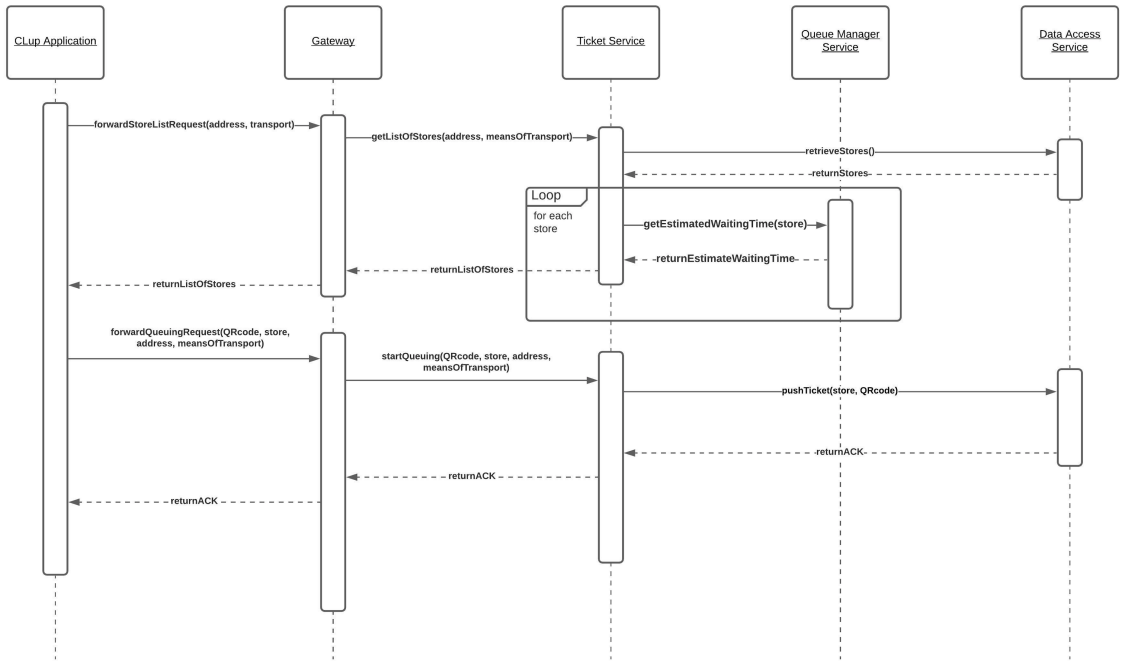
2.5.3 Manage entrances



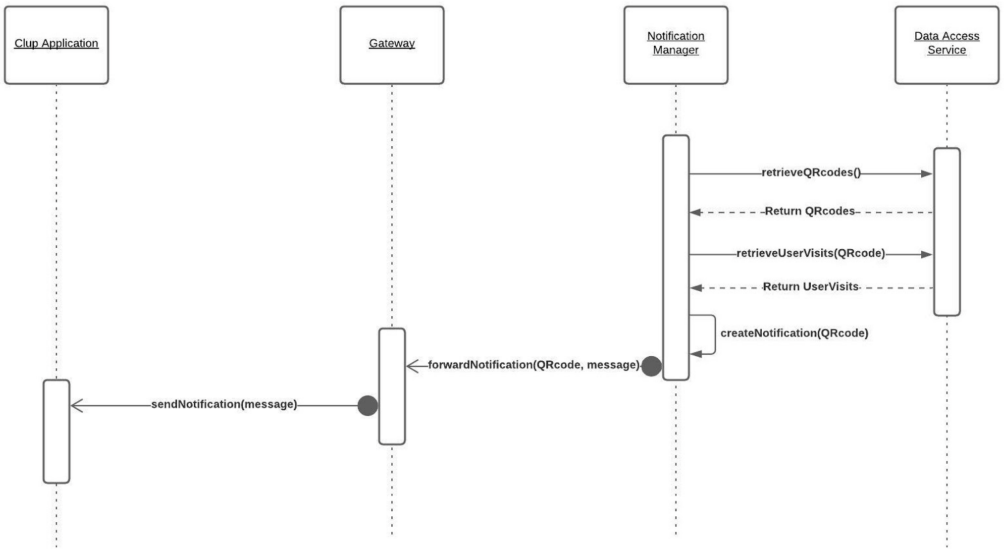
2.5.4 Book physically a visit



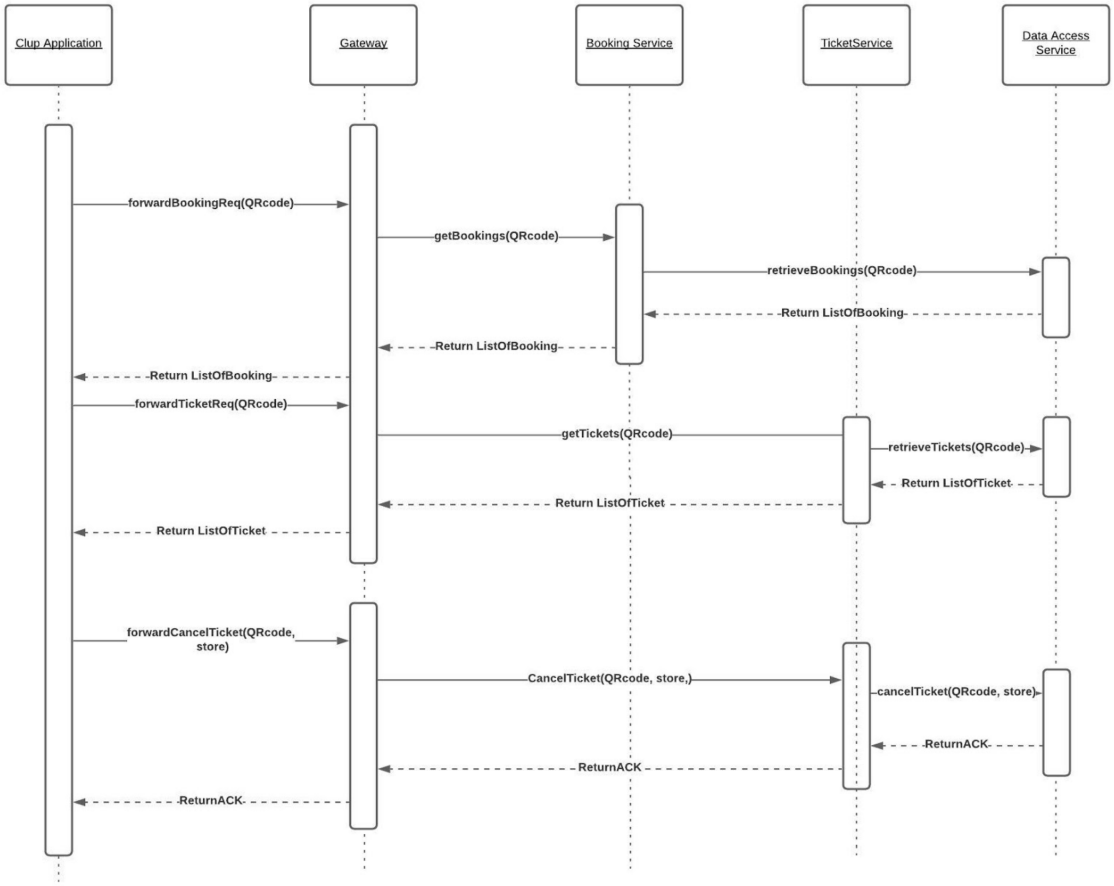
2.5.5 Take a ticket



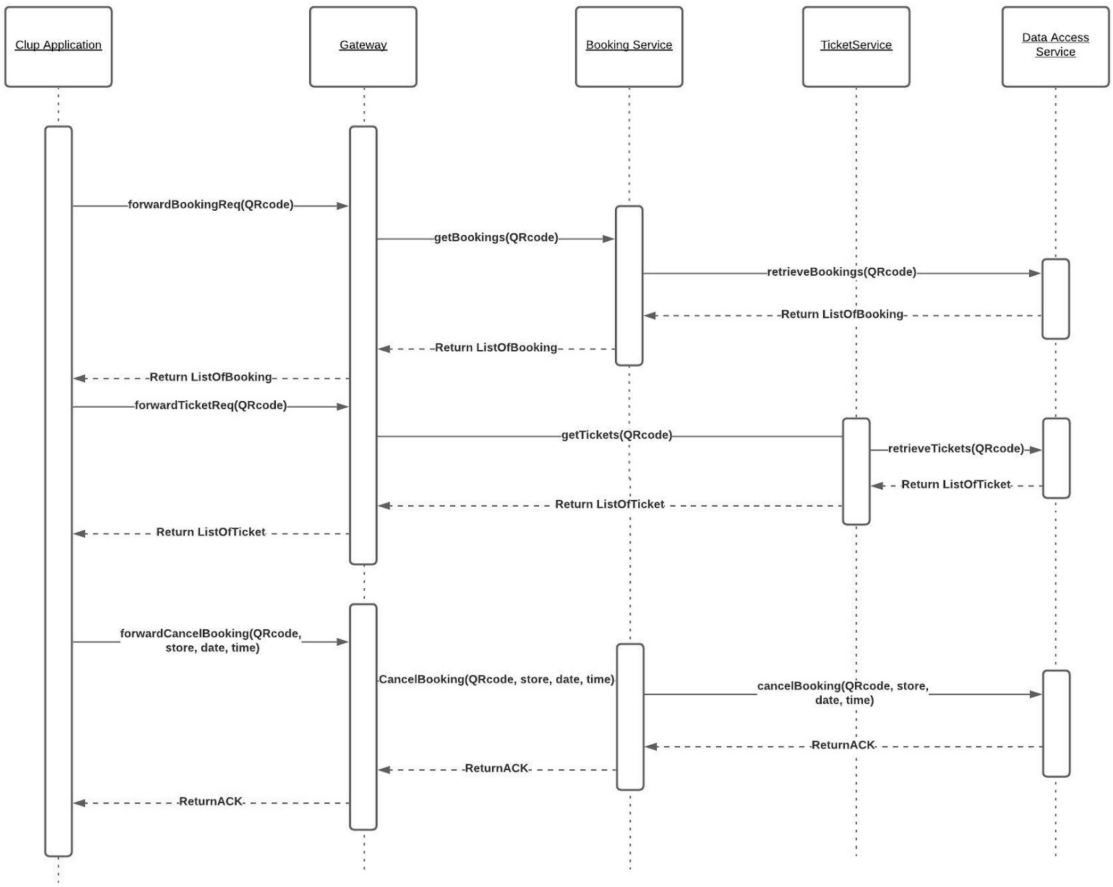
2.5.6 Receive a notification for a free slot to book



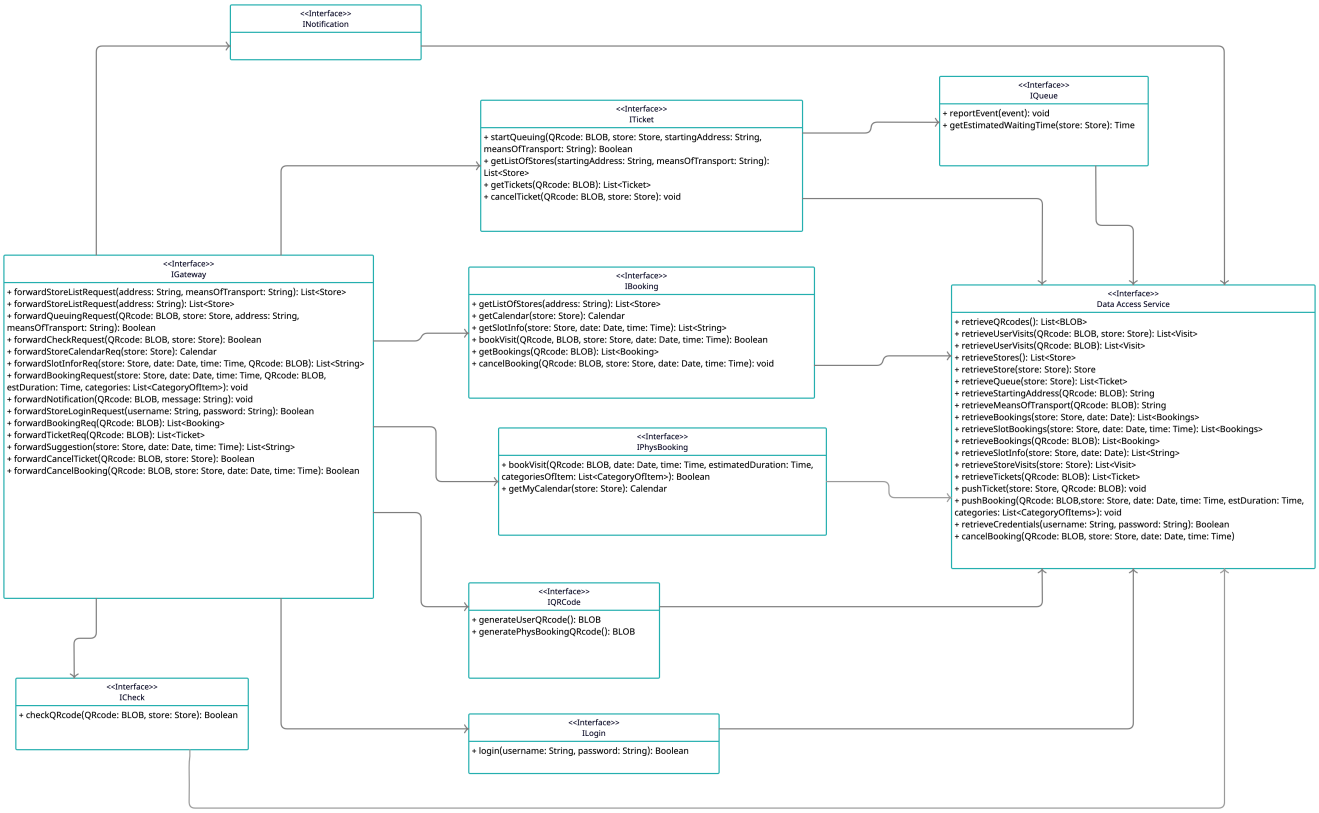
2.5.7 Cancel a ticket



2.5.8 Cancel a booking



2.6 Component interfaces



2.7 Selected architectural styles and patterns

2.7.1 Architectural style

As partly described in the overview, since the choice of the architectural style directly influences a number of QoS, a three-tier client-server architecture has been used to develop the entire system. This choice promotes a high decoupling of the system, increasing the reusability, scalability and flexibility. Furthermore, components in the application server have been thought to be cohesive and with low coupling among modules to make the system more comprehensible and modifiable.
The system does not embody a pure client-server paradigm. This is clear considering the inclination towards a data-centered model, since the data is centralized and accessed frequently by other components, which can modify data. Actually, it's neither a pure data-centered model because, even though all the components access a shared data structure, they aren't totally independent, in that, they don't interact only through the data store, but also between them, since each module is in charge to deal with a set of data which belong to its role. Pay attention to the fact that we're using stateless components, they completely rely on external persistent storage.

With respect to the database, instead, the application server acts as a client making queries and waiting for responses.

2.7.2 Communication protocols

The communication protocol used to exchange messages is HTTPS, which means we're using http over ssl. This is also endorsed by the crucial role that data plays in the system. In this way it is possible to reduce the coupling among client and server components. Besides, SSL is used to guarantee the security and the reliability of the connection.

Regarding the format in which the data are transmitted, JSON is used for its performances. It is less verbose and so more readable, since it is designed specifically for data interchange, unlike xml which can provide a lot more than data interchange. Besides, it is faster than xml by definition and allows a much faster parsing due to its lower complexity.

When dealing with Google maps API, the communication protocol used is RESTful: Google maps is indeed used as an external service and this protocol clearly suits this situation.

2.7.3 Patterns

Finally, to further decouple the presentation tier from the business tier a facade pattern has been used. This pattern is used under the shape of the "gateway" to provide an interface to the client through which it can access the system. A facade is an entity that serves as a front-facing interface masking a more complex underlying structure, realizing

the so-called “information hiding”. So that, in this pattern, the client interacts with a simpler interface instead of an intricate one thanks to the hiding of complexity provided by the gateway.

2.8 Selected architectural styles and patterns

2.8.1 Data Base

As written in the Deployment view, the database is SQL-like. Such decision is justified by the fact that we will have lots of requests from the database coming from different users or employees, so to cope in a proper way with this concurrency context we need to use the Transaction system typical of an SQL DBMS.

Since our system needs to handle dynamically the queues and these latters are saved in the database, the Queue Manager Service component, to be always updated on all changes in the queues, has to interact many times with the database.
The calls to the database are costly for our application, so in order to bring them to the minimum, a set of triggers are designed on the database to notify the Queue Manager Service every time a Create or Delete operation is executed on a Queue. Thereby, the calls to the database are optimized, since they are done only when an event occurs in a queue and are done only to the queue where the modification is done.

2.8.2 Algorithms

The scope of this section is to further increase the level of detail provided by this document. To do so, a brief, but detailed, description of how some algorithms work to provide some features is provided.

Queue management
The queue is “concretely” managed by the Data Base. Every time a new ticket is taken, and the “Data access service” becomes aware, it stores it in the data storage adding it at the end of the queue specifying in a boolean field whether or not it can be used to access the store. The first tickets, up to the maximum capacity of the store, are saved with true. When a ticket is true in this field, the correspective user is alerted by lowering his/her estimated waiting time until the minimum time necessary for the user to reach the store, if the estimated waiting time were lower, nothing is done. Once the maximum capacity is reached, all the new tickets are saved with false. A counter manages this. When someone enters the store an entrance time is registered. When someone exits the store, which means that an exit time is registered, the first of the tuple saved with this field as false, is updated, switching the value from false to true. The counter is decreased when someone exits the store and there’s no one else in the queue, since this counter counts the number of people that can enter the store and has as upper bound the maximum store capacity. When a ticket is cancelled, and the “data access service” becomes aware, the value of the field which specifies whether or not the ticket can be

used to access the store is read, if it's true, then the first of the tuple saved with this field as false, is updated switching the value from false to true. Then the tuple from the data storage is deleted, removing it also from the queue shifting the next positions forward.

Enhance store capacity
This algorithm starts when all the bookings for a specific time slot of a store(whether from the mobile app or at the store) are made, so when it becomes busy, and all the bookings comprise the category of items that the users intend to buy.
First, the Data access service invokes a method to pass all the categories of items of the bookings. for a specific time slot, and the maximum capacity of that store to the "queue manager service".
Second, the "queue manager service" checks if all the categories for such a specific time slot are different, allowing a maximum overlap of 2 stated categories for each type.
If the check is passed, then the "queue manager service" sends back to the Data access service a message to change the value of the boolean field, which specify whether or not a ticket can be used to access the store, in some tuple of the queue in the Data storage. The number of tuple changed is 10% of the maximum capacity of the store.

Estimated waiting time
It is calculated by the "queue manager service" when a user requests the list of stores to take a ticket. It uses the starting address and the means of transport used by the user. Every means of transport has an average velocity that is used to compute this calculation. When some tickets are added to the queue with "false" in the field which specify whether or not a ticket can be used to access the store, the calculation is computed also taking into account that there are other users before the new user in the queue. The first user that has "false" in such a field will have an estimated waiting time calculated summing: the time to reach the store with the stated means of transport, and the average time an user with a ticket spends inside the store, from this quantity is then subtracted the difference between the current time and the entrance time of the first user with the lowest entrance time of whom entrance time is not been used in other estimated waiting time. The second user will have an estimated waiting time calculated in the same way but using the entrance time of the second user with the lowest entrance time which hasn't been uses , and so on. If there are no more users with the entrance time, the new user will have an estimated waiting time calculated without taking into account the entrance time or the current time and then a message is sent to advise to stop decreasing the estimated waiting time of such new user until a new entrance time is registered in the Database. The estimated waiting time on the CLup application decreases, if the field is on true then the estimated waiting time reaches 0, otherwise it will remain at 1. When the estimated waiting time reaches 0, the CLup application waits 10 minutes. After that, it sends to the system a message to delete the ticket from the queue since it's expired.

Time statistics
The time statistics are used to infer the estimated duration of a visit of a long-term user when doing a booking he/she doesn't specify it. The "Time statistic service" calculates the average time a specific long-term user with a booking actually spends inside a store. It retrieves all the data about the duration of the booked visits of a user and calculates the average. This information is then saved in the data storage and used by the "queue manager service" when performing the "Dynamic managing of the queue".

Notification
The Notification Manager, every Monday, retrieves from the data storage all the data needed to create a custom notification for each long-term user.
For each QR code of the users, this component retrieves all the tickets, bookings and the store visited. Calculates which is the most visited store, summing the number of times a store appears in a user's visits, and the most frequent days and time at which this visits were done. Once these information are calculated, it checks if that time of that day is available, otherwise chooses the first available slot after this. Then it sends to the user a notification which suggests to book that slot.

3 User interface design

3.1 Home

Figure 1: homepage of the device with the CLup application downloaded

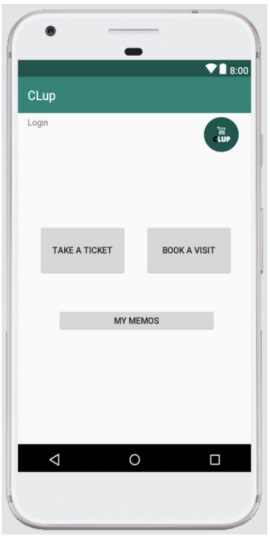


Figure 2: homepage of the CLup application

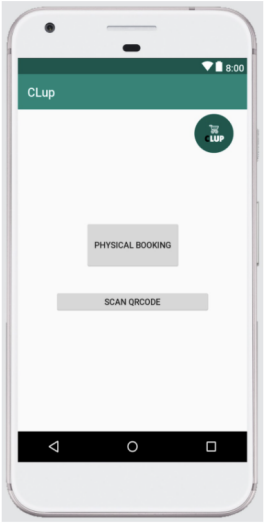


Figure 3: homepage of the CLup application after the login

3.2 Take a ticket

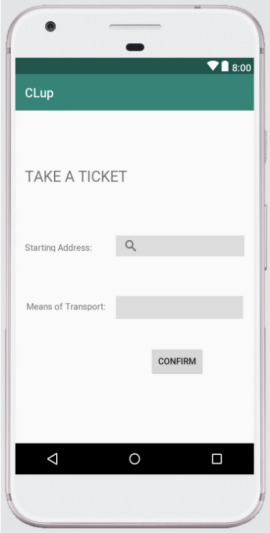


Figure 4: first page of the Take a ticket functionality (after have clicked on of the “Take a ticket“ button)

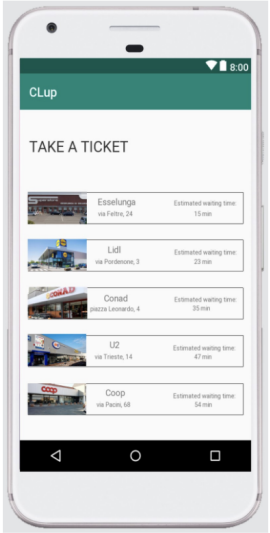


Figure 5: list of all the stores with their estimated waiting time

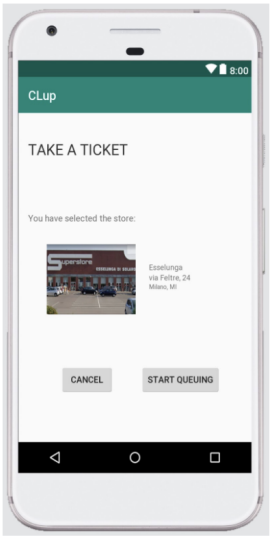


Figure 6: selection of a store

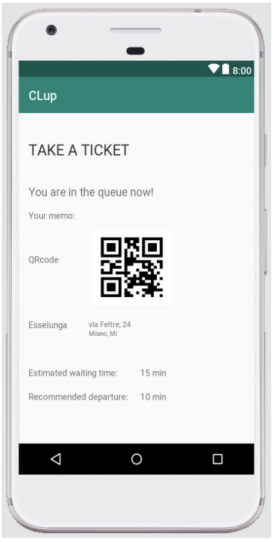


Figure 7: confirmation of the ticket taken and view of the memo

3.3 Book a visit

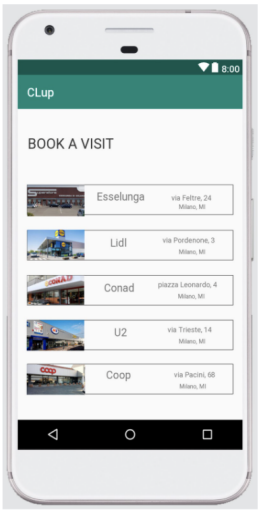


Figure 8: list of all the stores (after clicking on The “Book a visit” button)

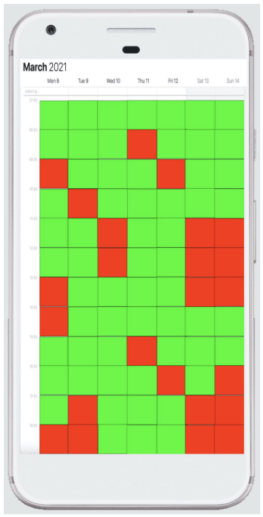


Figure 9: calendar of the store selected with free (green) and busy (red) slots

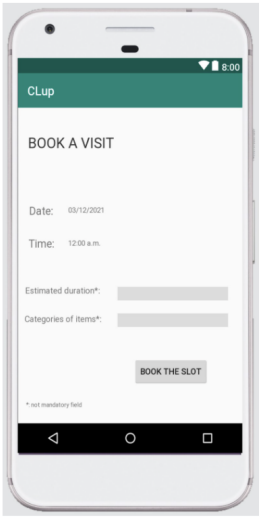


Figure 10: selection of a free slot

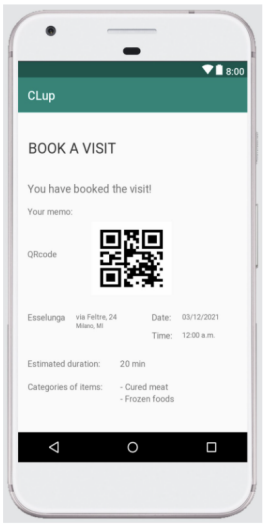


Figure 11: confirmation of the booking and view of the memo

3.4 Physical booking

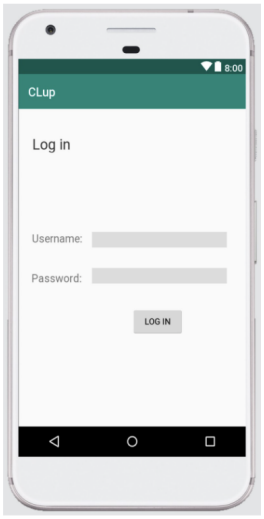


Figure 12: login form after clicking on the “Login” button

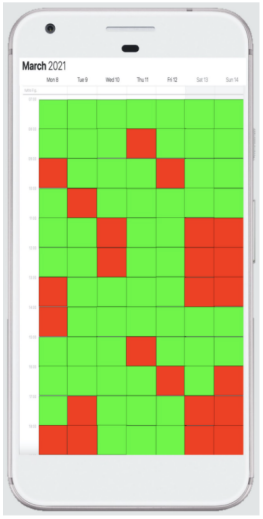


Figure 13: calendar of the store with free (green) and busy (red) slots (after clicking on the “Physical booking” button

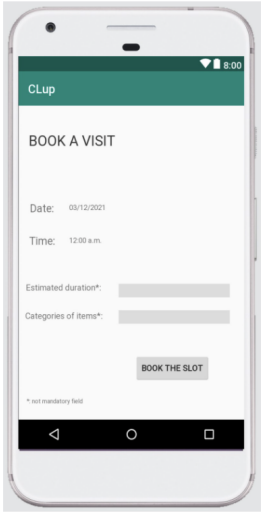


Figure 14: selection of a free slot

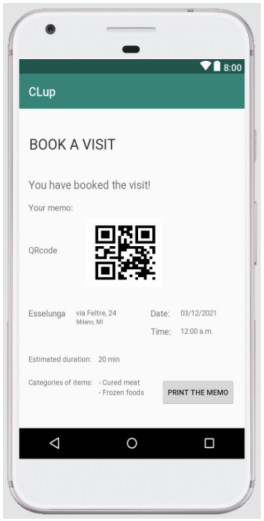


Figure 15: confirmation of the booking and view of the memo, with the possibility to print it

3.5 MyMemos

Figure 16: view of all memos of a user (after clicking on the “My memos” button)

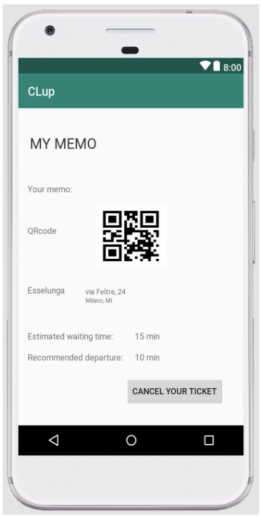


Figure 17: selection of a memo of a ticket with the possibility to cancel it

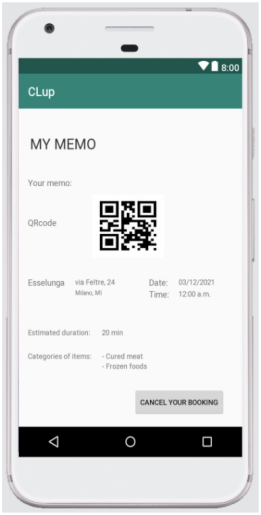


Figure 18: selection of a memo of a booking with the possibility to cancel it

4 Requirements traceability

1. The system provides, for every store, a calendar to book visits, also indicating whether a slot is available or not
Booking Service, Physical Booking Service, Data Access Service

2. User can do a booking for a visit in a free slot for a specific store;
CLup Application, Gateway, Booking Service, Data Access Service, Time Statistics Service

3. The System allows the user to specify the estimated duration of the booked visit;
CLup Application, Gateway, Booking Service, Data Access Service

4. User who have indicated their starting address are allowed to take a ticket;
CLup Application, Gateway, Ticket Service, Data Access Service, Google Maps Access Service

5. The System allows the user to specify categories of items he/she intends to buy during his visit;
CLup Application, Gateway, Booking Service, Data Access

6. The system creates a new unique QR code for each downloaded app;
CLup Application, QRCode Manager

7. The system saves the ticket taken by the user and associates it to the user's QR code;
CLup Application, Gateway, Ticket Service, Data Access Service, Queue Manager

8. The system saves the user's booking and associates it to the user's QR code;
CLup Application, Gateway, Booking Service, Data Access Service

9. User can take a ticket for a visit to a specific store;
CLup Application, Gateway, Ticket Service, Data Access Service, Google Maps Access Service, Queue Manager

10. The system provides an up-to-date estimated waiting time for the "take a ticket" functionality;
Queue Manager, Data Access Service, Ticket Service,Gateway, CLup Application

11. The CLup application manages and displays only the stores signed up in the CLup system;
CLup Application, Gateway, Booking Service, Ticket Service, Data Access Service

12. The system provides a sign-in function for the stores;
CLup Application, Gateway, Store Login Service, Data Access Service

13. The system should reserve 70% of the store's maximum capacity in a time slot for people that want to take a ticket;
Ticket Service, Data Access Service, Queue Manager

14. The system should reserve the 30% of the store's maximum capacity in a time slot for people that want to do a booking;
Booking Service, Data Access Service

15. The system should allow the stores to behave as if it were a mediator in the "book a visit" procedure;
CLup Application, Store Login Service, Physical Booking Service, QRcode Manager, Data Access Service

16. The system has to create temporary QR codes for physical tickets taken from the store's profile;
QRcode Manager

17. The system provides the printable version of the memo associated to the physical booking;
CLup Application

18. The system should allow the employee to scan the QR codes directly from his/her interface;
CLup Application

19. The system makes statistics on the time actually spent in the store by every long-term user;
Time Statistics Service, Data Access Service

20. The system can decide to enhance up to 10% the maximum capacity of the store in the same time slot when every customer inserts item's category he/she intends to buy;
Queue Manager, Data Access Service

21. The user can't take a ticket while the store is closed, that is outside of its working time;
Ticket Service

22. The user can't take more than one ticket for a specific store until the previous one is used or expired;
Ticket Service, Data Access Service

23. The system won't allow users to take a ticket for a store if the trip from the starting address to the store's address takes more than 45 minutes
CLup Application, Gateway, Ticket Service, Google Maps Access Service, Data Access Service

5 Implementation, integration and test plan

5.1 Overview

A key feature of the system architecture and project plan is the fact that the integration test plan drives and is driven by the integration plan.
Since it is almost impossible to develop error free software and, quoting Dijkstra, program testing can be used only to show the presence of bugs, but never to show their absence, the phase of verification and validation takes a crucial role, in particular the necessity of having a plan to integrate components while testing them. Thus, for this reason, the aim is to find as many bugs as possible until the release date of the application.

5.2 Implementation Plan

The entire system, with its relative sub-components, has to be implemented, but also tested and integrated exploiting a bottom-up approach. Using bottom-up allows to integrate the system in an incremental way, so that, also the testing can proceed in parallel with the integration, in particular the unit testing of each single component, to test sections of code. This is fundamental given the fact that an incremental integration facilitates bug tracking and it's a quite simple approach with respect to the benefits it brings.
The bottom-up approach requires that the implementation has to be done gradually from the lower components up to the top, starting from the internal ones up to those that interact with the client side.
There are some components that rely on some others so a priority among the components is present. For example, every component described in the Component diagram interacts with the "Data access service".

Then, the first step will be to implement the "Data access service", which is the component implementing all methods that allow access to the Database and perform operations on it.
The second step will be to implement the "QR code manager", since it's used by other components and it doesn't call any other component.
The third step will be to implement the "Store manager" and it's relative subcomponents, since it's needed to guarantee at the store, through a login, the operations to manage the entrances.
The fourth step will be to implement the "Efficiency Manager", this component with it's relative subcomponents performs a crucial role since it's often called by invocations that come from the "Data access service" that are triggered by other components.
The fifth step will be to implement the "Google Maps access service", since it will be needed by the visit manager component.
The sixth step will be to implement the "Visit Manager", since all it's components require other components that are already deployed at this moment. The seventh step will be to implement the "Notification Manager". There isn't an actual reason to implement this

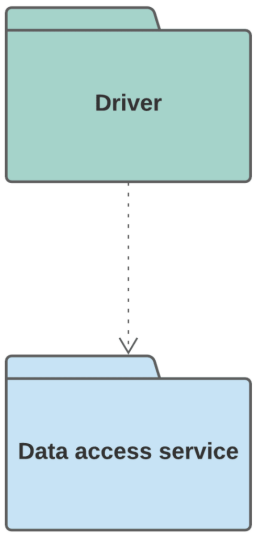
component only at this point, it can be implemented in parallel with any other previous component.
The last component that will be implemented is the "Gateway" since it's needed only to dispatch messages to other components when dealing with the Client side.
The order in which we will implement our components in the application server is then:

1. Data access service
2. QR code manager
3. Store Manager
4. Efficiency Manager
5. Google Maps access service
6. Visit Manager
7. Notification manager
8. Gateway

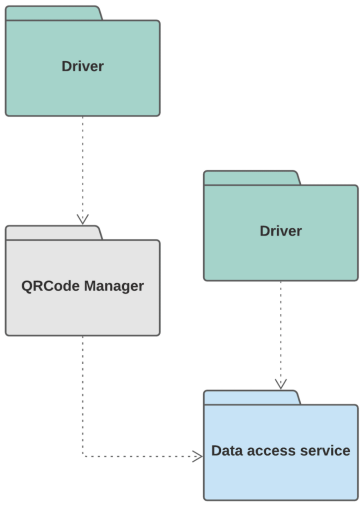
5.3 Integation strategy

To implement and test the different functionalities of the system a bottom-up approach has been used. The following diagrams describe how the process of implementation and integration testing takes place, according to a bottom-up approach.

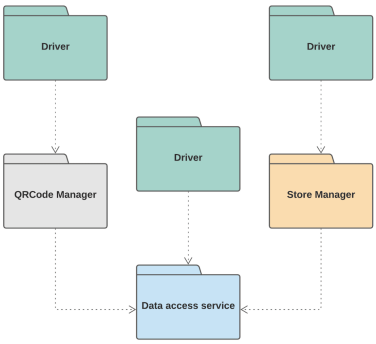
(1) At first, the "Data access service" is deployed and unit tested checking if it's able to communicate with the database correctly.



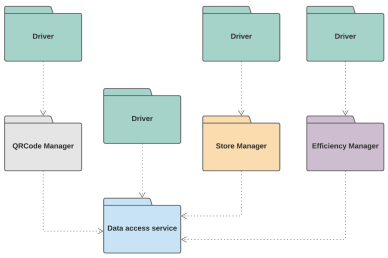
(2) Then, to provide a correct identification of the tuple in the database, the "QRCode Manager" is implemented and unit tested using a driver to check whether it creates a new unique QR code when a new mobile app is downloaded or when a customer does a physical booking at the store.



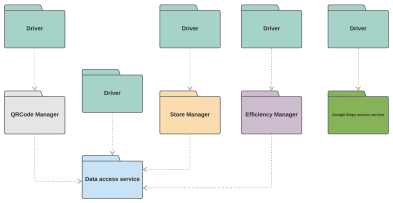
(3) Then, to provide the functionalities of: allowing an employee to sign into the application, so that he/she can scan QR codes and do physical bookings, and to check the validity of a memo, by crossing the scanned QR code with the information retrieved from the data storage, the "Store Manager" component is implemented. It is unit tested through the usage of a driver that checks whether these functionalities are provided correctly.



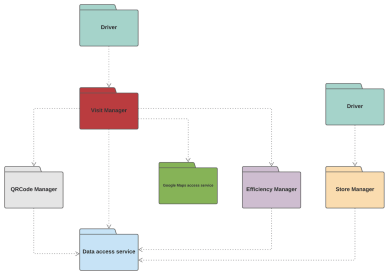
(4) Then is implemented the "Efficiency manager". It's needed to implement this component at this point since it's used by other components to provide many crucial functionalities. There's no need to develop a driver, since it can be unit tested directly using the Data access service.



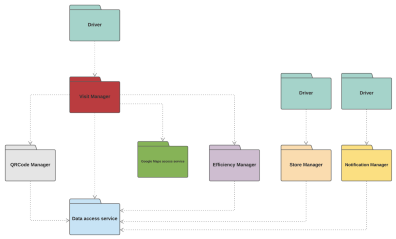
(5) Then it's the turn of the "Google Maps access service". This component is deployed and unit tested through the usage of a driver to check whether it connects to and retrieves the correct data from the Google Maps API.



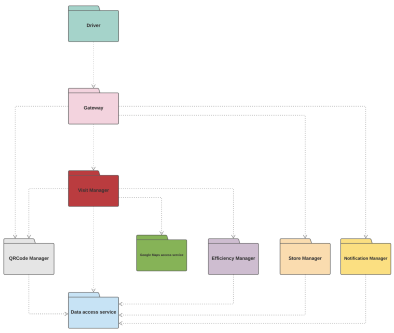
(6) Then is implemented the "Visit Manger". It's notable the choice of implementing this module at this point since all the other underlying components are already implemented. All the functionalities provided by this component can be unit tested, through the usage of a driver, which starts the procedures to perform the features provided by this component, acting almost as if it were an actual client.



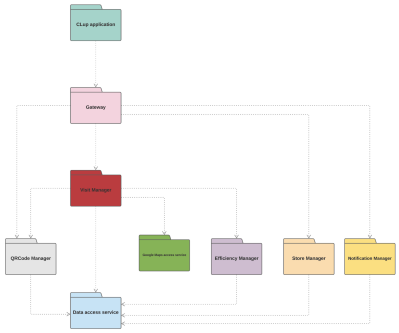
(7) Then the "Notification Manager" is deployed. It has been decided to implement it at this point because it might be more meaningful to unit testing it once the underlying infrastructure is already implemented.



(8) Then the different drivers from the previous steps are substituted by the "Gateway", which is unit tested using a driver that acts as if it were a client that performs some actions.



(9) Finally the remaining part, characterizing the client side, is implemented. Actually this last step can be performed in parallel to any other step, since the architectural choices made allowed to decouple the client side from the server side, besides the client side is realized by the drivers until the "Gateway", which allows to connect the CLup application to the application server, is implemented.



Once the system is fully implemented and the integration testing is over, the time of the system testing will come, since it's needed to conduct tests on the completed and integrated system.

5.4 System Testing

Once the System is completely integrated, and the integration testing it must be tested as a whole to verify that functional and non-functional requirements are satisfied. Moreover, the testing environment should be as close as possible to the production environment. The purpose of this test is to identify weaknesses of the system and whether the performance of the system are acceptable.
The system testing can be divided in more types, each of this is performed in a different moment to obtain a full tested system.

- Functional testing: verifies if the application satisfies the functional requirements described in the RASD
- Performance testing: identifies the presence of inefficient algorithms, query optimization possibilities or hardware/network issues highlighting possible bottlenecks. It establishes also a performance baseline that can be used to be compared with different versions of the CLup system
- 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 System Testing

In this final sub-chapter are explained further information regarding the verification and validation process.
Before the unit-testing, the integration testing and the system testing, should be done also the analysis activities that require a fully inspection activity at all stages of the development process together with an automated static analysis.

In particular, the inspection starts before the writing of the code and concerns requirements and design specified by RASD and DD. This is particularly important because building the right product implies creating the correct requirements specifications that contains the needs and realize the goals of the software product. If such documents are wrong or incomplete, the developers won't be able to write what the stakeholders wanted. For this reason, it's fundamental that a quality team starts with an analysis of RASD and DD documents as soon as a first version of such documents is released.
The analysis activity, furthermore, can be divided in review between peers and inspection. The first should be organized by the developer team with appropriate people in order to present the product step-by-step to the members of the meeting that comment on the correctness of the product. It is an informal procedure through which an in-depth examination is performed.
While, the second should be done by the quality team, which is a team of trained and professional inspectors. Periodically, the quality team is subjected to have "inspection meetings". During those meetings a member of the development team (called "reader"

in this case) reads the document, paraphrasing part by part the documents, while the inspector points out what they have noticed as problems in the code. Quality team is not in charge of fixing anything, that is something that must be accomplished by the authors.
Changes made by the authors are checked to make sure that they are correct and that problems have been fixed. This activity of inspection is really important since it is proved that it is cost-effective.

6 Effort spent

In this section the tables of each team member will follow, indicating their work done in the project. It should be taken into account that of the hours declared, 35 hours were carried out in a group and therefore are counted in each table

Matteo Makovec

TOPIC	HOURS
Initial discussion	7 hours
Introduction	1 hour
Architectural Design	28 hours
User Interface Design	5 hours
Requirements Traceability	5 hours
Implementation, Integration and Test Plan	4 hours
Discussion with the Tutor	2 hours
Revision after the discussion with the tutor	5 hours
Final Revision	3 hours
Latex document composition	3 hours

Lorenzo Male

TOPIC	HOURS
Initial discussion	7 hours
Introduction	1 hour
Architectural Design	27 hours
User Interface Design	1 hour
Requirements Traceability	6 hours
Implementation, Integration and Test Plan	4 hours
Discussion with the Tutor	2 hours
Revision after the discussion with the tutor	7 hours
Final Revision	3 hours

Gabriele Morelli

TOPIC	HOURS
Initial discussion	7 hours
Introduction	1 hour
Architectural Design	26 hours
User Interface Design	1 hour
Requirements Traceability	5 hours
Implementation, Integration and Test Plan	13 hours
Discussion with the Tutor	2 hours
Revision after the discussion with the tutor	5 hours
Final Revision	3 hours

7 References

Specification document: "R&DD Assignment AY 2020-2021.pdf"
RASD: "RASD2.pdf"