POLITECNICO DI MILANO 1863

SOFTWARE ENGINEERING 2 PROJECT

DESIGN DOCUMENT (DD)

CLup

Version 1.0

Authors
SAMAN FEKRI
PARNIYA SAEEDZADEH

Supervisors
Prof. ELISABETTA DI NITTO
Prof. MATTEO ROSSI

Click **here** for Github repository

January 10, 2021

Contents

List of Figures

1 Introduction

1.1 Purpose

This is the Design Document (DD) of CLup application. The purpose of this document is to discuss more technical aspects regarding architectural and design choices that must be made, so as to follow well-oriented implementation and testing processes. This document is to provide more technical and detailed information about the software discussed in the RASD document.
In this DD we present hardware and software architecture of the system in terms of components and interactions among those components. Furthermore, this document describes a set of design characteristics required for the implementation by introducing constraints and quality attributes. It also gives a detailed presentation of the implementation plan, integration plan and the testing plan.
In this document we mostly talk about features that listed below:

- The High-level architecture of the system.
- Main components of the system.
- Interfaces provided by the components.
- Design patterns.

Stakeholders are invited to read this document in order to understand the characteristics of the project being aware of the choices that have been made to offer all the functionalities also satisfying the quality requirements.

1.2 Scope

The software wants give them the possibility to line up for a super market and notify them when it's their turn.

- **basic function:** Allow application to retrieve a number for users to be in the line waiting for their numbers to be close. Then they will go and stand in the line but in a way that they wont waste their time standing in a queue for long time
- **advanced function 1:** The second functionality point out about booking a visit from application for supermarket which is similar to booking a visit for museums but has some differences. This feature let application for customer to indicate the approximate expected duration of the visit. Alternatively, for long-term customers, this time could be inferred by the system based on an analysis of the previous visits. The application might also allow users to indicate, if not the exact list of items that they intend to purchase, the categories of items that they intend to buy. This would allow the application to plan visits in a finer way, for example allowing more people in the store, if it knows that they are going to buy

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Manager	A person who manages the shop
User	A regular citizen who wants to shop
Clerk	A person who works for a specific shop
Lineup	An imaginary queue of current person who wants go to shop
Ticket Machine	A stand that clerk can get and print ticket
Scanner	A device that scans QR code
QR Code	is a type of matrix barcode (or two-dimensional barcode)

1.3.2 Acronyms

DD	Design Document
UI	User Interface
RPC	Remote procedure call
HTTP	Hypertext Transfer Protocol
REST	Representation State Transfer
JSON	JavaScript Object Notation
RASD	Requirement Analysis and Specification Document
GPS	Global Positioning System
app	Application
API	Application Programming Interface
QR Code	Quick Responsible code
MVC	Model, View, Controller

1.3.3 Abbreviations

BS	Basic Service of CLup
AF1	Advance Function 1 of CLup
AF2	Advance Function 2 of CLup
Rn	Requirement number n

1.4 Revision history

Date	Modifications
10/01/2021	First version

1.5 Reference Documents

- Specification Document: "R&DD Assignment A.Y. 2020-2021.pdf"
- Slides of the lectures.

1.6 Document Structure

This document is divided in seven sections.

- **Chapter 1** describes the scope and purpose of the DD, including the structure of the document and the set of definitions, acronyms and abbreviations used.
- **Chapter 2** contains the architectural design choice, it includes all the components, the interfaces, the technologies (both hardware and software) used for the development of the application. It also includes the main functions of the interfaces and the processes in which they are utilised (Runtime view and component interfaces). Finally, there is the explanation of the architectural patterns chosen with the other design decisions.
- **Chapter 3** shows how the user interface should be on the mobile and web application.
- **Chapter 4** describes the connection between the RASD and the DD, showing the matching between requirements described previously with the elements which compose the architecture of the application.
- **Chapter 5** traces a plan for the development of components to maximize the efficiency of the developer team and the quality controls team. It is divided in two sections: implementation and integration. It also includes the testing strategy.
- **Chapter 6** shows the effort spent for each member of the group.
- **Chapter 7** include the reference documents.

2 Overall Description

2.1 Overview

The application architecture has three logic layer that describes below:

- **Presentation layer (P)** The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user.

- **Application layer (A)** is the heart of the application. This layer contains the functional business logic which drives an application's core capabilities. the layer where all your business logic resides is the application layer.

- **Data access layer (D)** comprises of the database/data storage system and data access layer. It picks up useful information for the users in the database and passes them along the other layers.

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability. As mentioned, modularizing different tiers of an application gives development teams the ability to develop and enhance a product with greater speed than developing a singular code base because a specific layer can be upgraded with minimal impact on the other layers. It can also help improve development efficiency by allowing teams to focus on their core competencies. Many development teams have separate developers who specialize in front- end, server back-end, and data back-end development, by modularizing these parts of an application you no longer have to rely on full stack developers and can better utilize the specialties of each team.

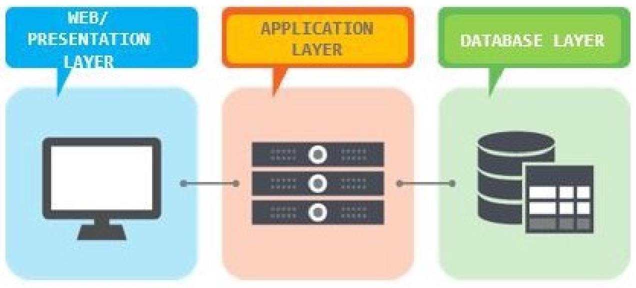


Figure 1. *Three-tier architecture*

the chief benefit of three-tier architecture its logical and physical separation of functionality. Each tier can run on a separate operating system and server platform - e.g., web server, application server, database server - that best fits its functional requirements. And each tier runs on at least one dedicated server hardware or virtual server, so the services of each tier can be customized and optimized without impact the other tiers. some benefits of three tier development are:

- Faster development
- Improved scalability
- Improved reliability
- Improved security

2.2 Components view

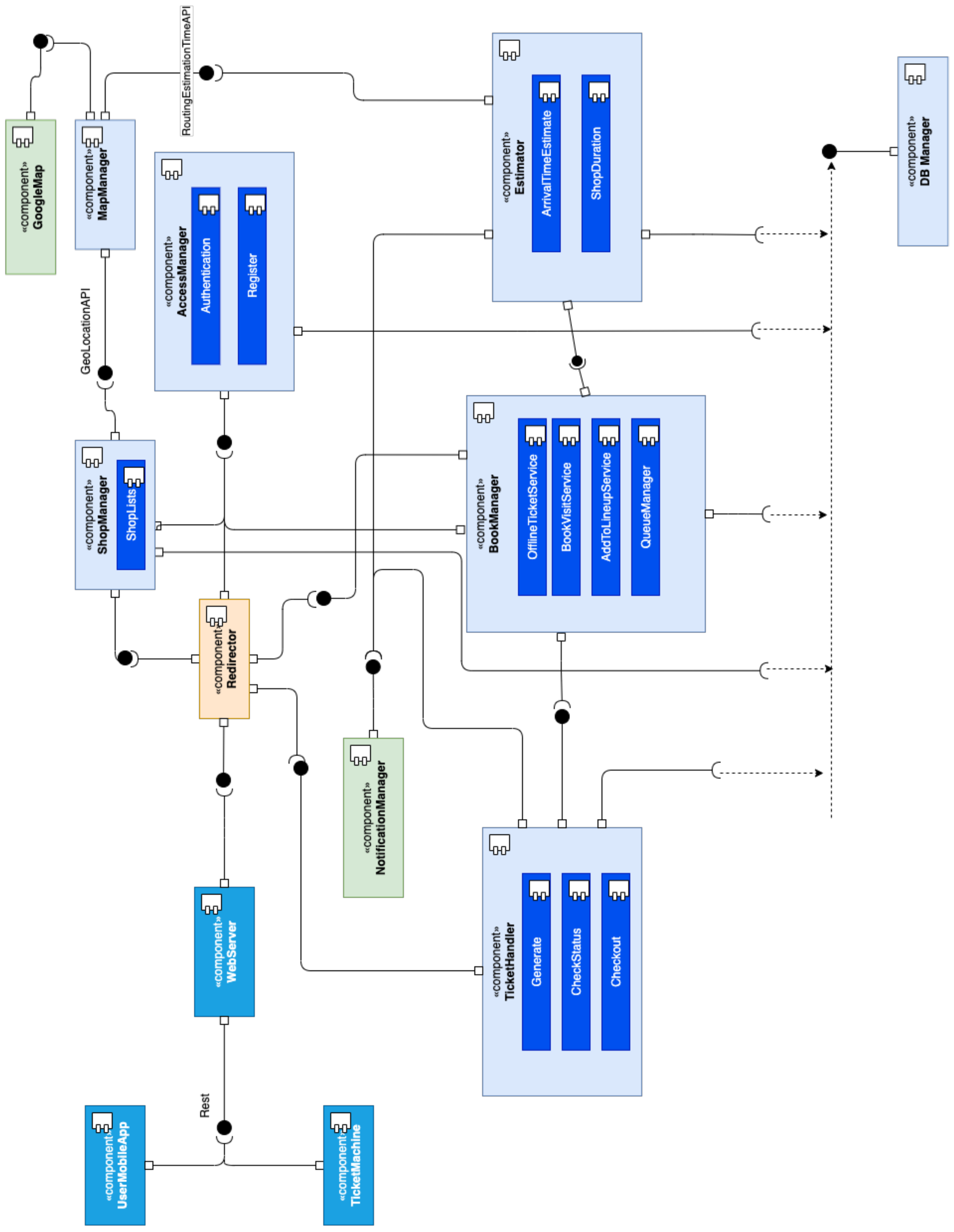


Figure 2. *Component Diagram*

The following component diagram gives a specific view of the system focusing on the representation of the internal structure of the application server, showing how its components interact. The application server contains the business logic of our software. Other elements in the diagram, besides the application server, have been depicted in a simpler way just to show how the communication is structured among these components and the application server.

- **Notification server:** it manages the notification which is used to send to users when its going to be close to be their turn and also their turn to enter the supermarket so it is connected to Estimator component.

- **Booking manager:** it is divided into some sections:
  - **offlineTicketService:** which it handles and print the tickets in person for whom they do not have smartphone.

  - **BookVisitService:** which is a booking for those who wants to take a visit in the supermarket and wants to book it before.

  - **AddToLineupService:** which is a service that users can reserve a slot for themselves at that moment by generating a QR code.

  - **QueueManager:** This module handles the interactions with queue for each shop.
- **Estimator:** this component estimates the time which is divided into two section:
  - **Arrival Time Estimation:** It is going to estimate the time takes from the current location to the store. This component use MapManager which is connected to the google map API. This component ask from notification manager to send a notification to user in proper time.

  - **Shop Duration:** The other section is the shop duration in which it estimates the duration of shopping for each customer thus the system can estimate the time for other users entrance. It also connects to google map component to recognize the user's location to notify him if he is nearby the supermarket.
- **TicketHandler:** its also divided into some sections:
  - **Generate:** which is about generating QR code for the use of customers to make line-up or booking.

  - **CheckStatus:** which is anout scanning customers QR code to check if it is valid or not.

  - **Checkout:** it manages the scanning of QR codes which it results to understanding the checking out of customers and it helps to improve the estimation time for other.

- **ShopManager:** this component is related to the shops. users could see the near shops through this components and manager who can add shop into the system and it connects to google map which is a element for them to add the location of the shop.

- **DBMSManager:** this is the component that allows every other component in the system to interact with the database. The Interface provided by this component contains all useful methods to store, retrieve, update data into the database from different actors. Every internal component of the application server uses some methods of its interface.

- **Redirector:** this component simply dispatches the requests and calls to methods from the users and the managers to the core of the application server. Every method is redirected to the proper component that can handle it. Also, responses and data sent back pass through this component to reach the applicant.

- **MapManager:** its connected to google map and make our specific interface and in case if we want to use other map services, we should just change this module.

The external component of the system is also mentioned:

- **GoogleMap:** this is the component that provides a useful interface used to visualize the map of a specific location requested by the user/managers and estimate time between two points.

2.2.1 Additional specification

In this project we use fat client for some reasons, first the clients use their cellphones and nowadays cellphones have very good computation power besides, mobiles have their own view that we cannot build in the server and then send to them. usually apps use framework which use webview are slower than normal apps. In addition, we prefer fat client computers over thin clients because fat clients allow easy customization and greater control over installed programs and system configuration. Because output is locally generated, a fat client also enables a more sophisticated graphical user interface and reduced server load.

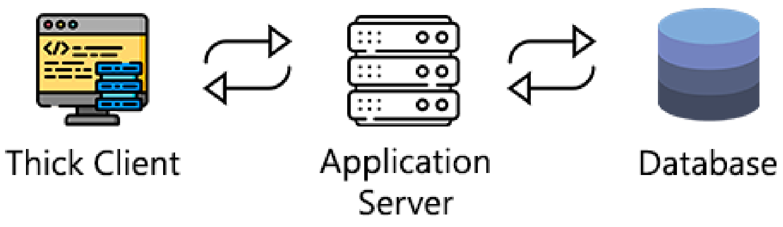


Figure 3. *Thick(Fat) client*

In this project, we could duplicate the redirector because as you can see in the component diagram the redirector could be bottleneck of the system. besides, we could have two redirectors at the peak of usage and in the night we could shutdown one of them.
We could duplicate the application server then use redirector component as a load balancer between them to guarantee the reliability of the application.

2.3 Deployment view

In the Deployment diagram in the figure are shown the most important components.

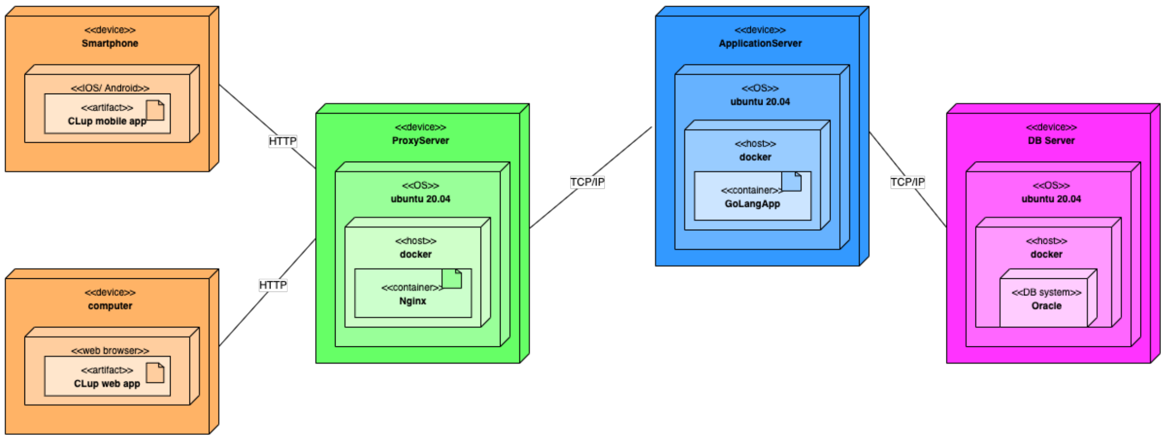


Figure 4. *Deployment Diagram*

The deployment has some devices:

- **Smartphone:** the user will use, which can be a smartphone or a tablet using as operating system either iOS or Android. The execution environment is the built CLup app.

- **Computer:** the clerk use this to put the offline user in the line and it also has scanner to scan user ticket in entrance and checkout from the shop.

- **Proxy Server:** in this device we use nginx as a reverse proxy server. A reverse proxy accepts a request from a client, forwards it to a server that can fulfill it, and returns the server's response to the client.

- **Application Server:** It is supposed to be a dedicated server running a linux distribution specific for server use. As an example of OS we choose Ubuntu. Other distribution can be used like Red Hat Enterprise Linux, Debian, Open-SUSE. As execution environment we use GoLanng because it is fast and suitable for microservice.

- **DB Server:** It consists in another server where we run the DB system Oracle. We choose to run the database in a separate server and not in the same as the Application Server in order to increase scalability.

As you can see in the diagram, we install docker in all of our server to increase portability and easy deployment of the servers.

2.4 Runtime View

2.4.1 Login

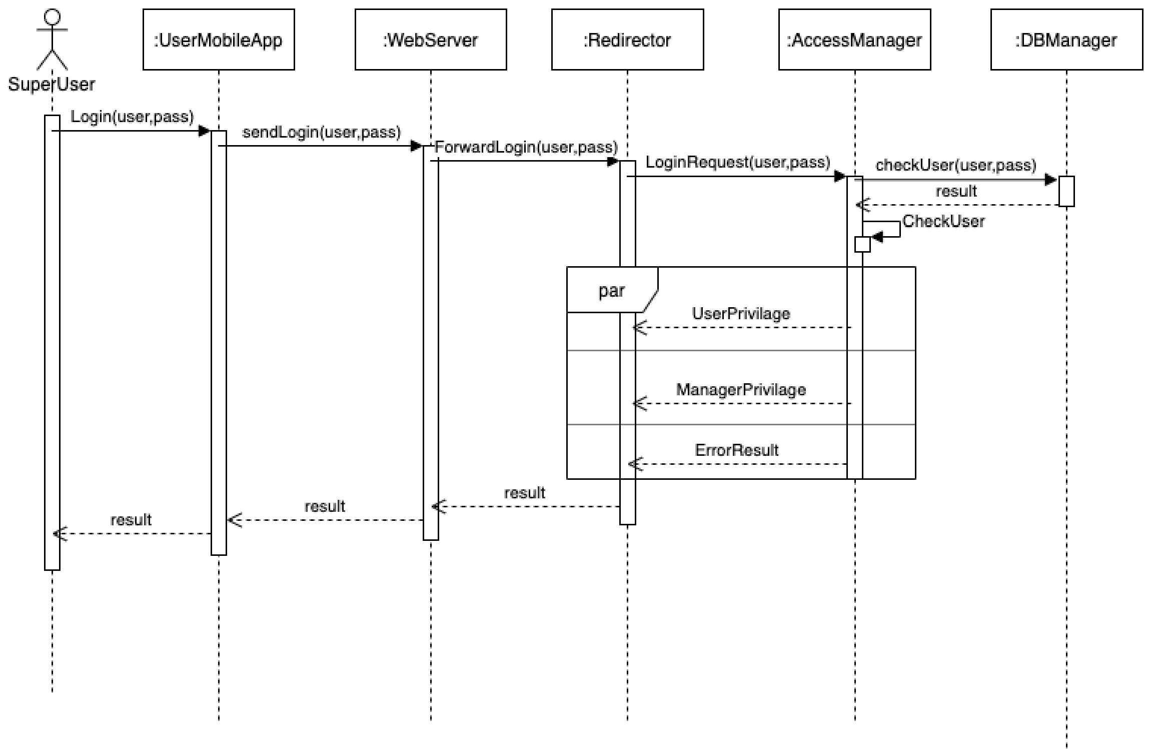


Figure 5. *login sequence diagram*

In this sequence diagram is described the login process by a user. first of all,the user login to the application through entering username and password. then it is sent to web server and after that it is forwarded to Redirector. through Redirector component, these username and passwords are sent to the right component which it is AccessManager and requested to this component by Redirector. Thus, in this step the AccessManager component checks if the user and passwords are for either users or managers or if they enter them wrong.after checking users by AccessManager, they will be sent to DBManager and forwarded to the user app, through the Redirector.

2.4.2 GetShopList

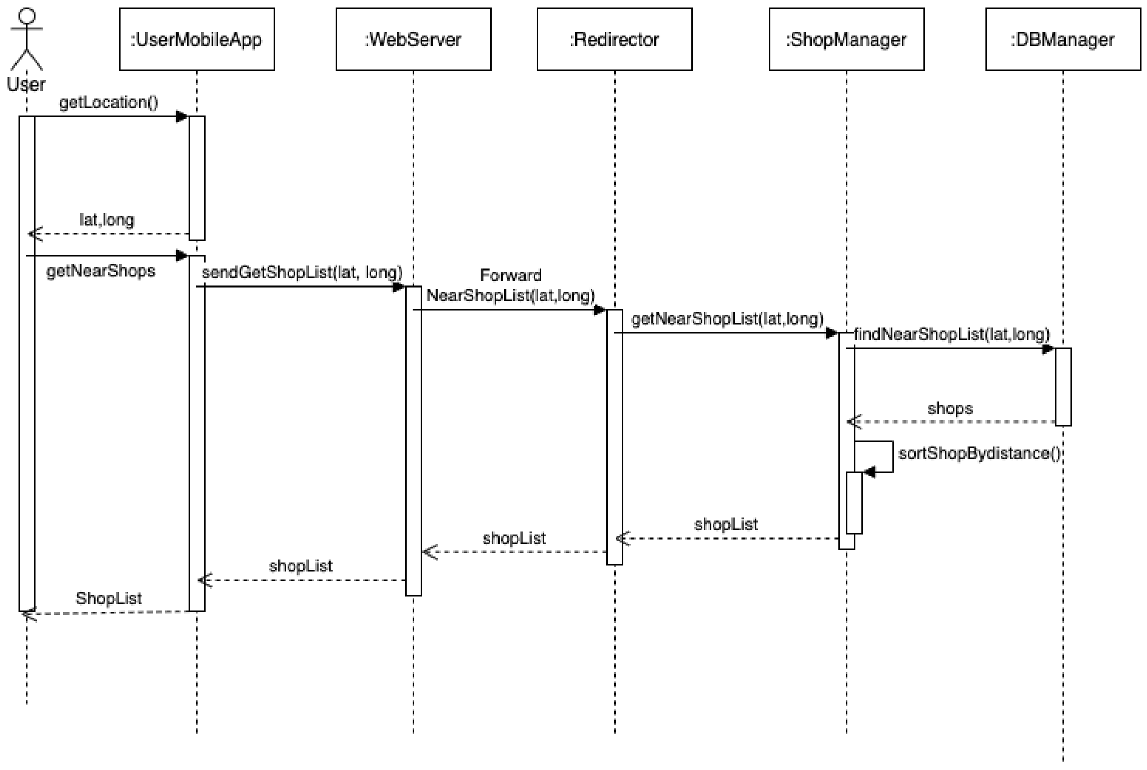


Figure 6. *GetShopList sequence diagram*

In this diagram, The user gets his location from the application, then through the UserMobileApp the shop lists are sent to the web server. Then the Redirector recognize that what component should do this so it will forward them to the ShopManager. As long as the shopManager gets the shop lists, through the DBManager it finds out the near shop lists and after sorting based on the distance of users, they will forwarded to the user through Redirector.

2.4.3 AddToLineUp

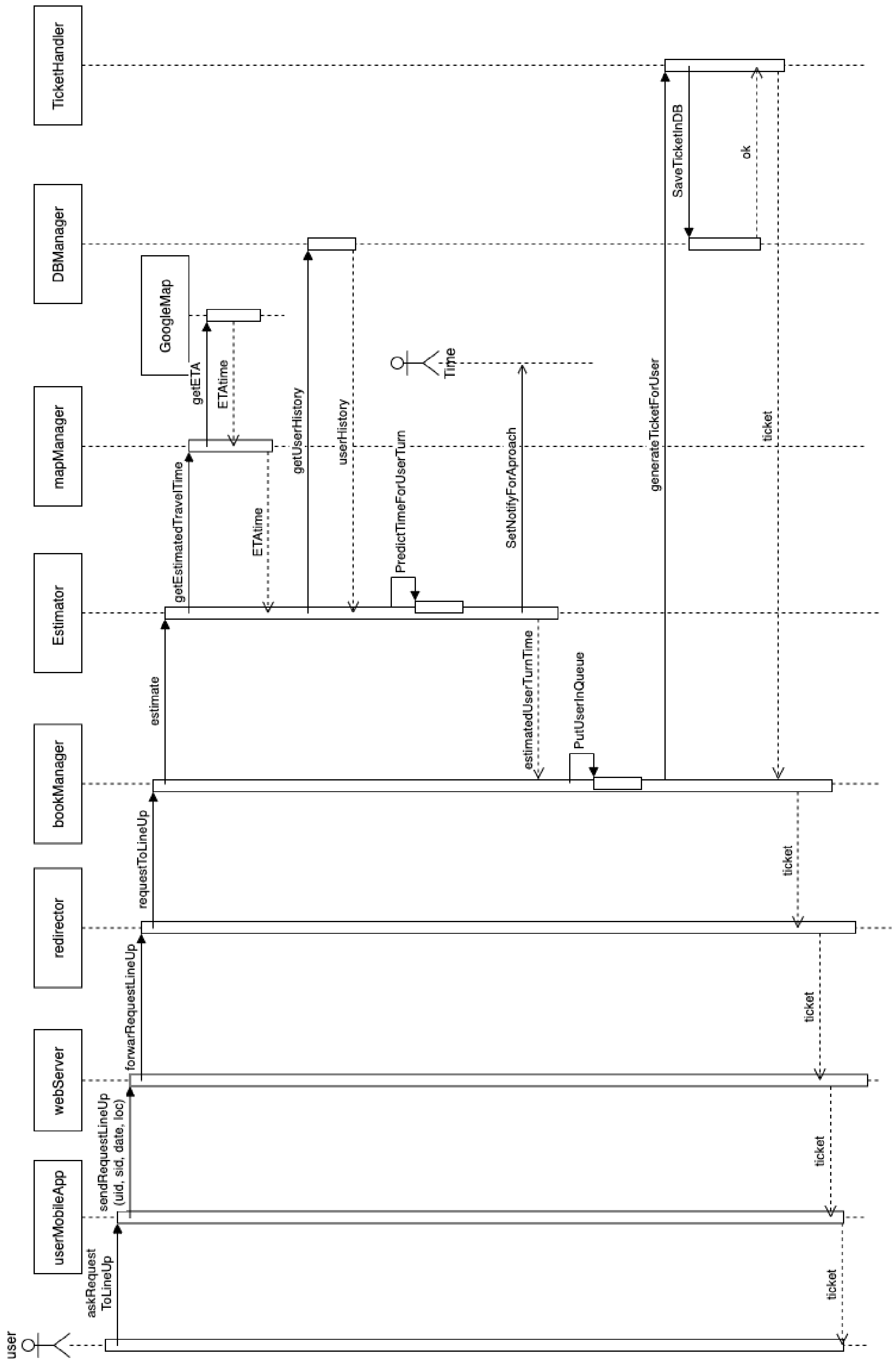


Figure 7. AddToLineUp sequence diagram

In this diagram,firstly the user asks to request for lining up, then like the previous diagram,it sends the request to web server and through Redirector, it is chosen to send the request to bookManager. Then, Estimator component get estimated time of arrival from the mapManger to manage the location of the user and within using that it can estimate the proper time. Meanwhile the mapManager use google map for this part. Also the Estimator set a notification to send it to the users. After all the bookManager generate a ticket for users by TicketHandler and thus forward to the user.

For the two next diagram its almost the same, just the concept is a little bit different, which for the second one is booking a visit with the same procedure and the third one is exactly like the first one except the process id offline in the third one.

2.4.4 BookAVisit

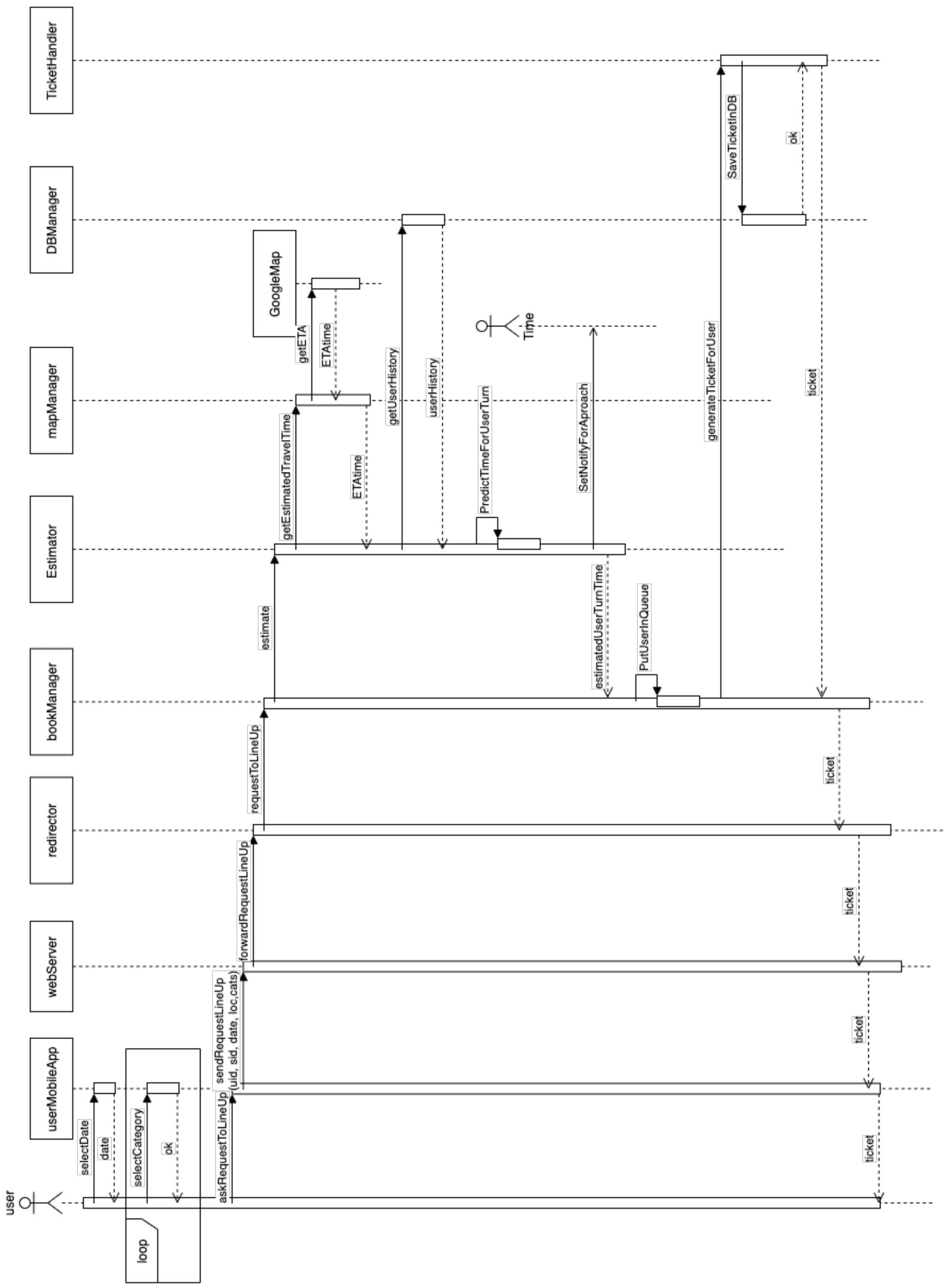


Figure 8. *BookAVisit sequence diagram*

2.4.5 OfflineLineUp

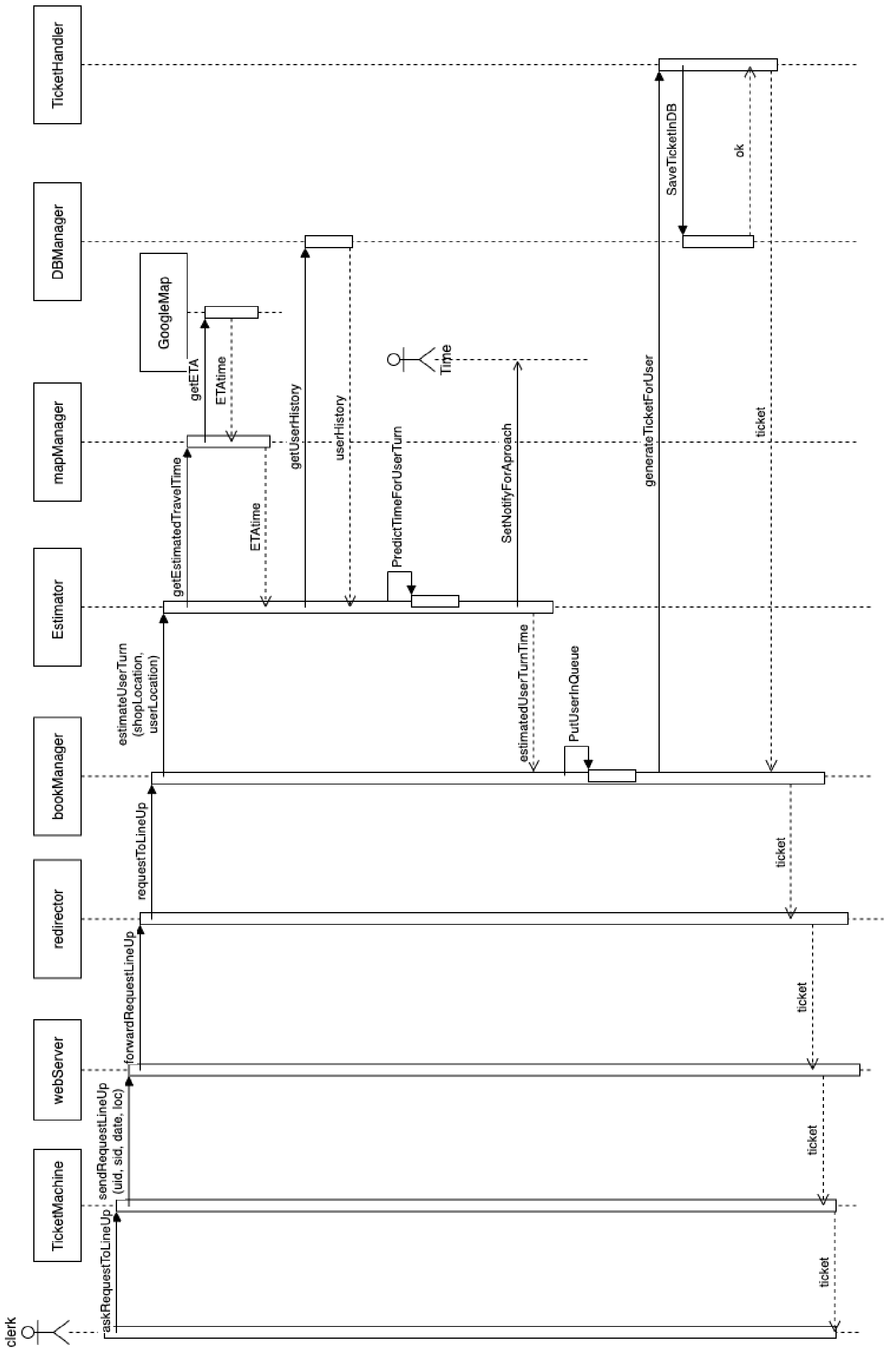


Figure 9. OfflineLineUp sequence diagram

2.4.6 Enter

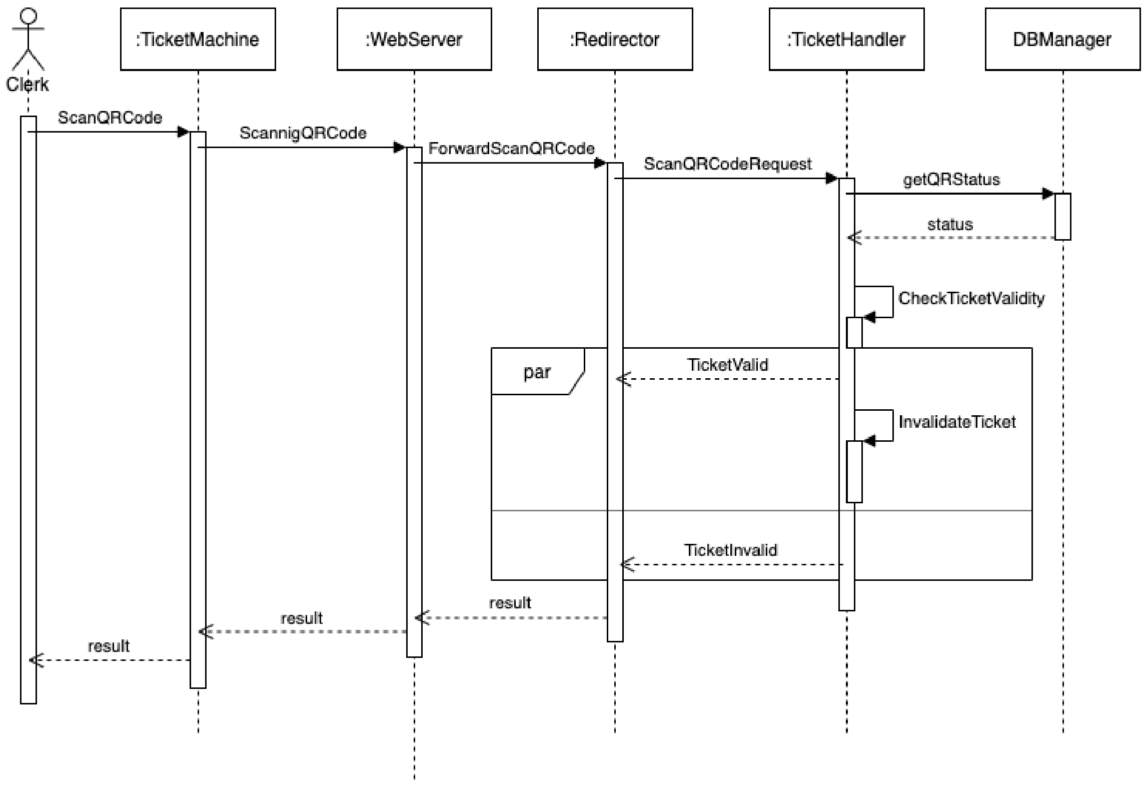


Figure 10. *Enter sequence diagram*

In this diagram, the clerk scan the QR codes through ticketMachine. It then goes through web server and through director it is forwarded to tickrtHandler and will check if its valid or not. Thus after getting by DBManager, the result is going to be forwarded to clerk.

2.4.7 Checkout

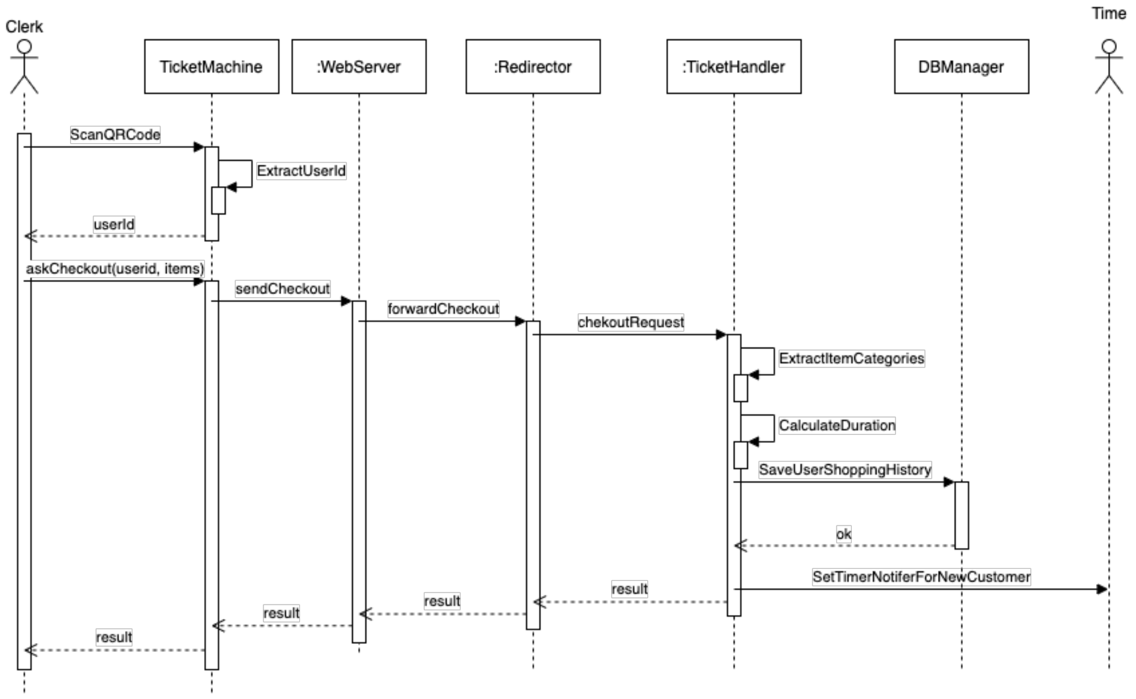


Figure 11. *Checkout sequence diagram*

For checking out diagram, its almost like the previous one, except some differences. One of them is that the ticketHandler recognize the duration of shopping and will calculate it and also extract the item categories chosen by users.Moreover the ticket-Machine also extract users id at the beginning.

2.5 Component Interfaces

In the next diagram are described the main methods which can be invoked on the interfaces and their interactions, referring to the most important processes reported in the runtime view section.

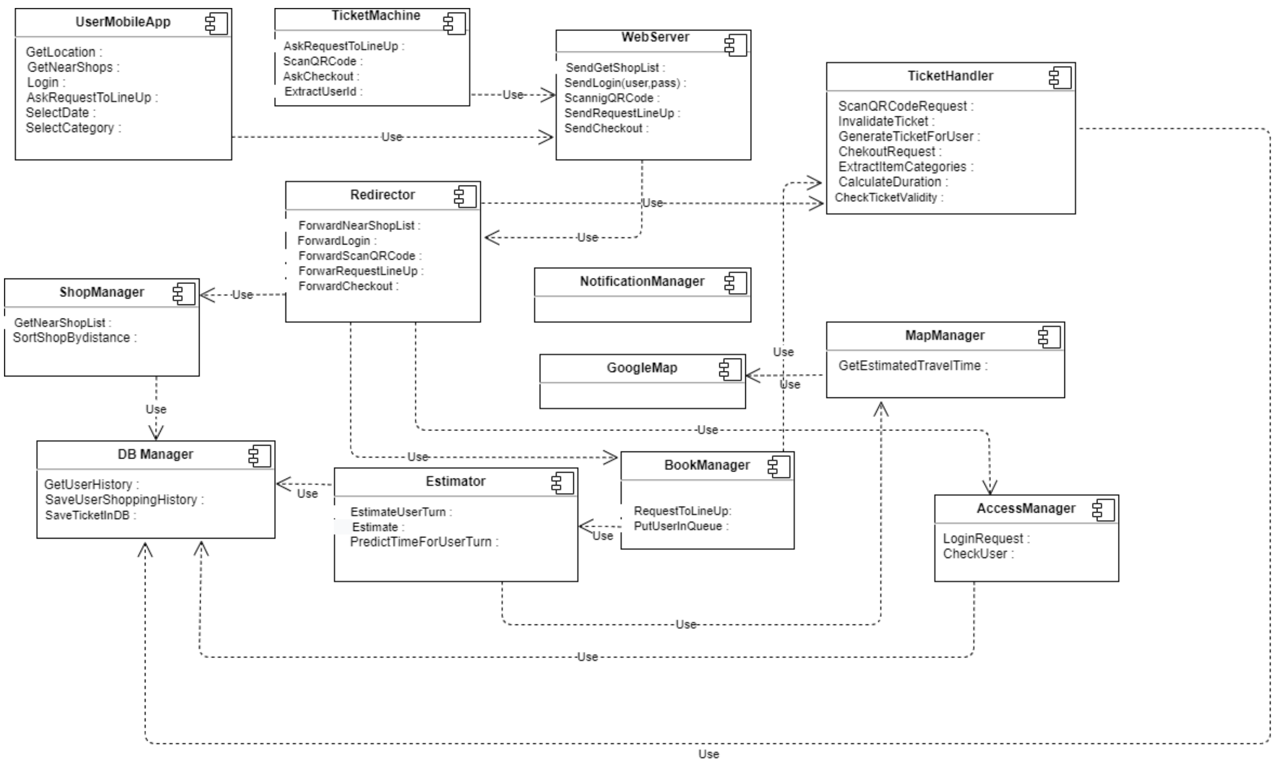


Figure 12. *Component Interface Diagram*

2.6 Selected Architectural styles and patterns

Our system is based on a client/server architecture with three layers. Presentation, application and data access are physically separated. We choose this kind of architecture in order to have a modular system, also we use fat client to reduce the server load and smartphones nowadays are very powerful so, the costs of server will be less. Moreover we separated physically the database from the application server, to increase scalability and security, and it's also possible to use a distributed database. This choices promote a major 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 communication between the Mobile application and the Server will be done via HTTP requests following REST principles. The communication between modules inside the application server and DB manager is RPC. When we use RPC, the programmer can use procedure call semantics and writing distributed applications is simplified because RPC hides all of the network code into stub functions.

In addition, we use some caches in the client side so we avoid some interactions between the client and server so we could reduce server loads. For example, in a short period of time users location do not change so, we could cache shops near the user for short amount of time and do not send requests to server.

We use JSON format for the communication data because, JSON uses less data overall, so you reduce the cost and increase the parsing speed. Readable: The JSON structure is straightforward and readable. You have an easier time mapping to domain objects, no matter what programming language you're working with.

The model we choose for this project is Model View Controller (MVC). This model improves the reusability and maintainability of code. One of the most important feature of this design pattern is separation of concerns.

- **View** is that part of the application that represents the presentation of data. Views are created by the data collected from the model data. A view requests the model to give information so that it resents the output presentation to the user.

- **Controller** is that part of the application that handles the user interaction.
- **Model** component stores data and its related logic. It represents data that is being transferred between controller components or any other related business logic.

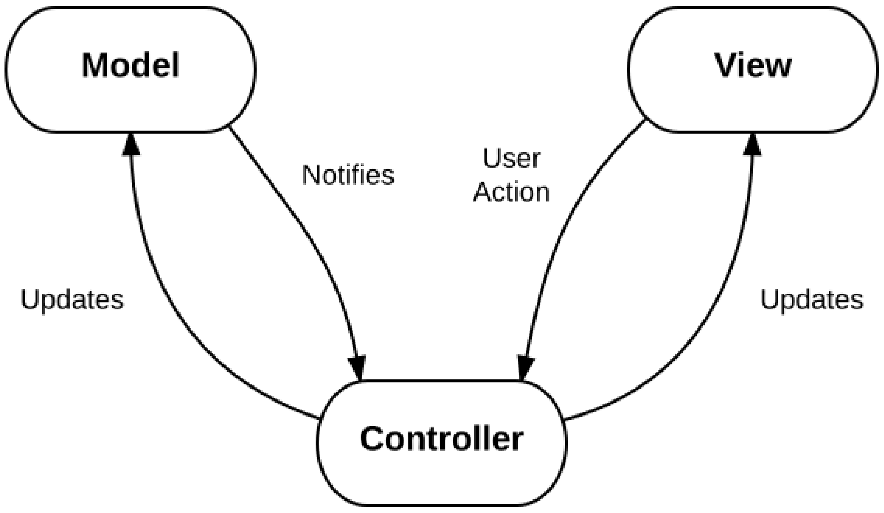


Figure 13. *Model View Controller*

2.7 Other Design Decision

In this section, we elaborate some decision we take.

- **Flutter:** Flutter is a free, open-source mobile SDK that can be used to create native-looking Android and iOS apps from the same code base. Flutter helps app developers build cross platform apps faster by using a single programming language, Dart, an object-oriented, class defined programming language. We chose to use this Framework in order to have a mobile app that works both with iOS and Android, with performances similar to the native development for each platform, but only having to write the code once. In Flutter, every piece of the user interface is a widget: like text, buttons, check boxes, images. There are also "container widgets" that contain other widgets. Widgets can be stateless, which are immutable, or stateful, which have a mutable state and are used when we are describing a part of the user interface that can change dynamically.
  Lastly we want to remember that Flutter is not the only framework that can be used to build cross-platform mobile apps, another option can be React Native, based on JavaScript. We decided to use Flutter because of it's advantages in comparison with React Native and other UI software development kit, which are: better performances and increasing popularity due to it's simplicity of develop.

- **Golang:** Go is a statically typed, compiled programming language. Go has the same performance as C, but it is much easier to maintain than Java. Without the need for a virtual machine, Go boasts easier maintenance and no warming up period. These and many other characteristics are what make Golang stand out from its competitors.

- **Docker:** Containers work a little like VMs, but in a far more specific and granular way. They isolate a single application and its dependencies—all of the external software libraries the app requires to run—both from the underlying operating system and from other containers. All of the containerized apps share a single, common operating system (either Linux or Windows), but they are

compartmentalized from one another and from the system at large.
The benefits of docker:

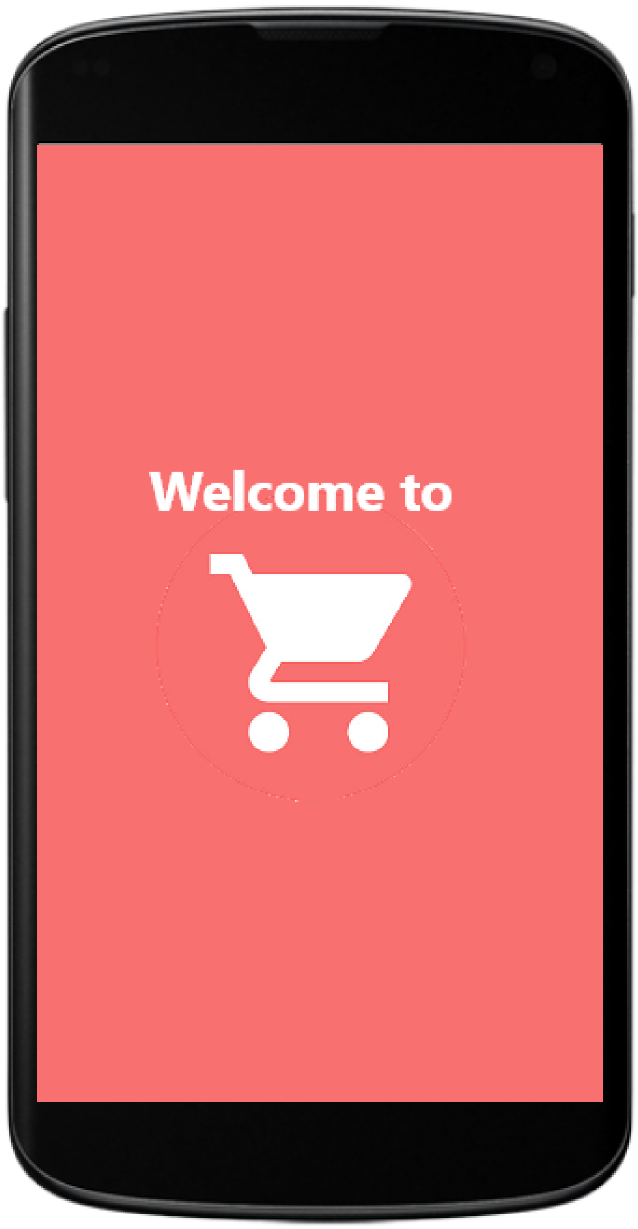
- Docker enables more efficient use of system resources.
- Docker enables faster software delivery cycles.
- Docker enables application portability.
- Docker shines for microservices architecture

3 User Interface Design

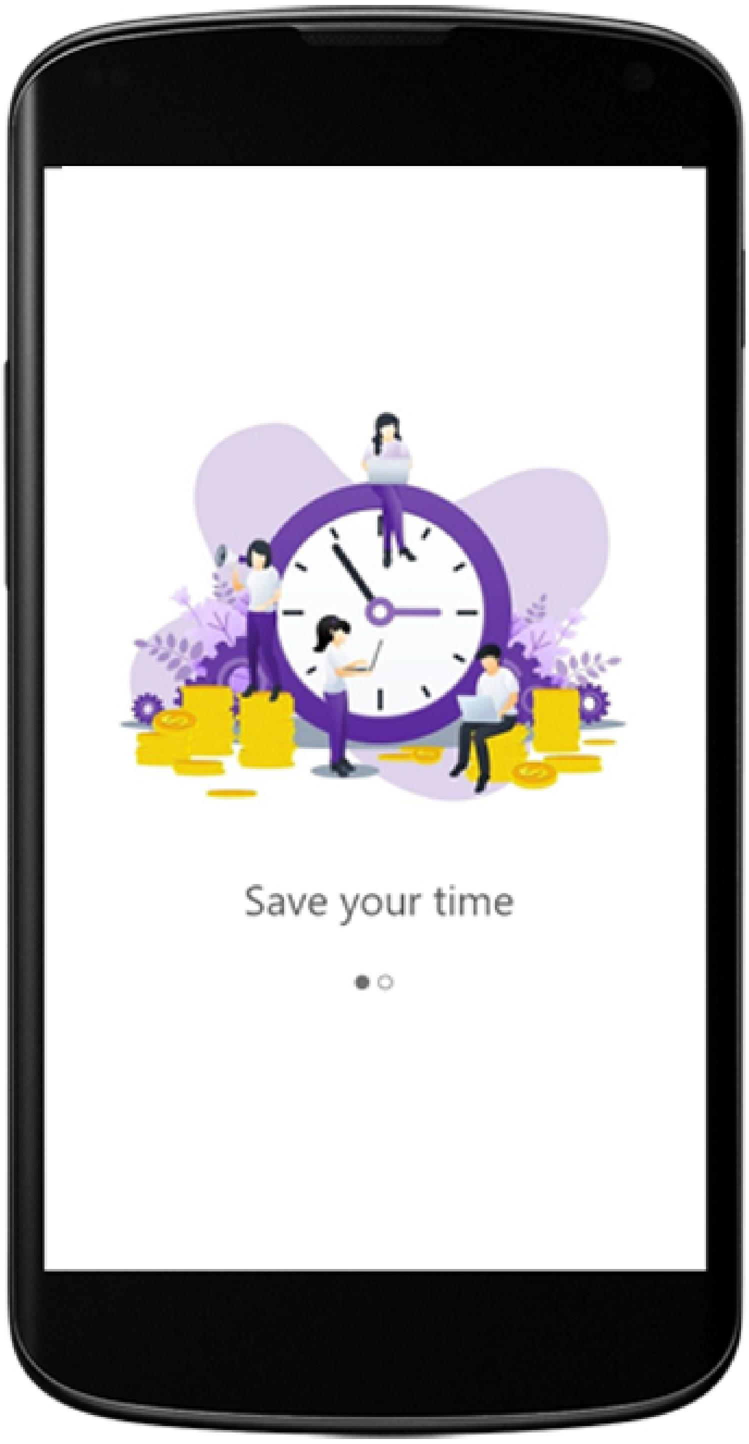
3.1 Mockups

the following mockups show how it should be the look of the mobile application used by a user(3.1.1 to 3.1.21) and the aspect of the application used by managers (3.1.22) to (3.1.26)

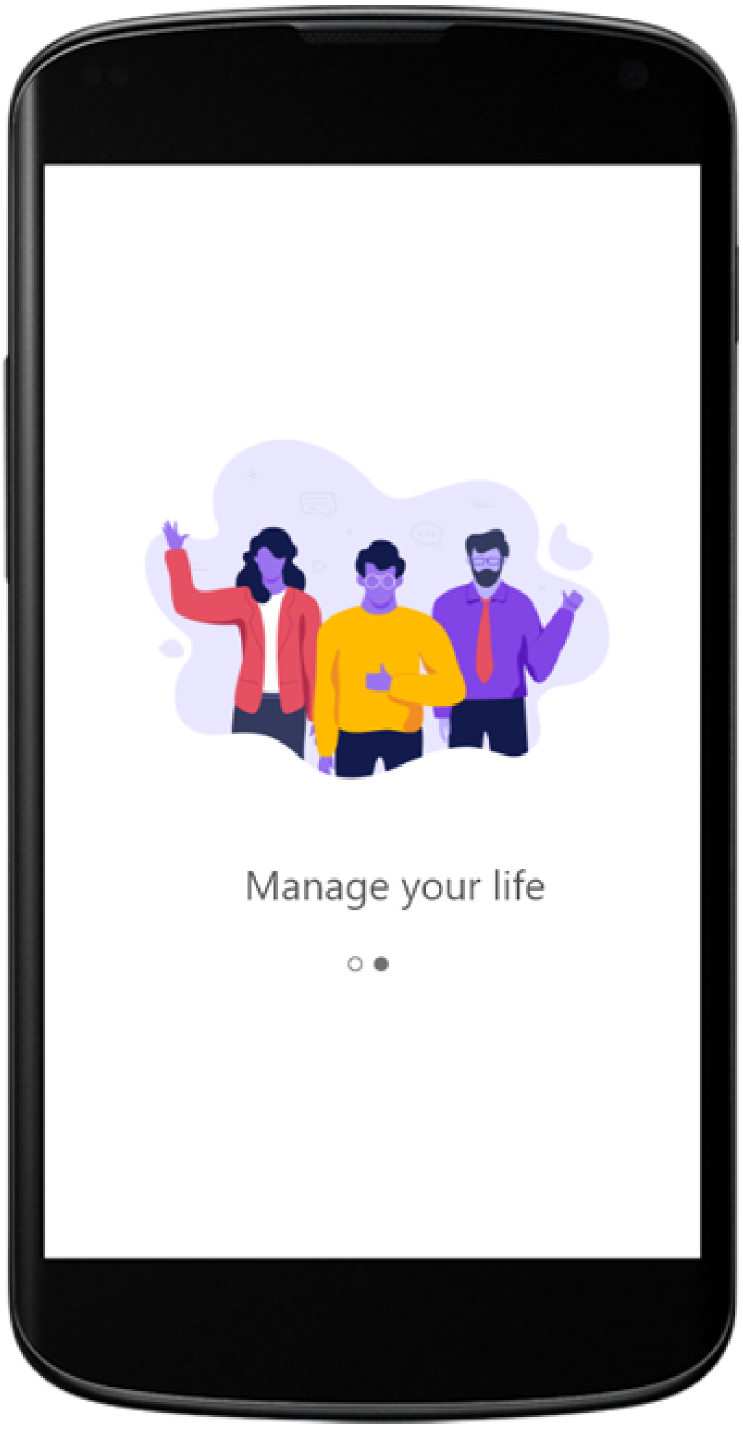
3.1.1 Logo



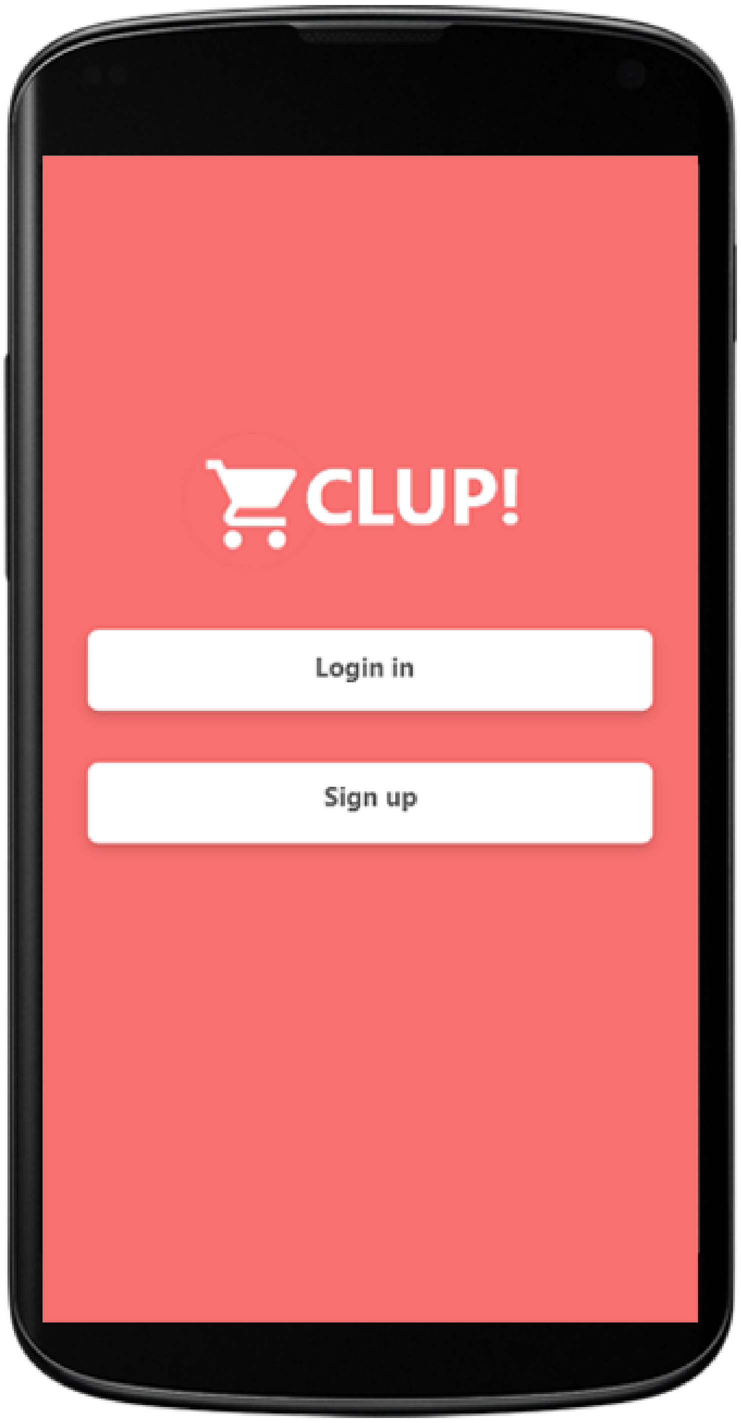
3.1.2 Splash Screen



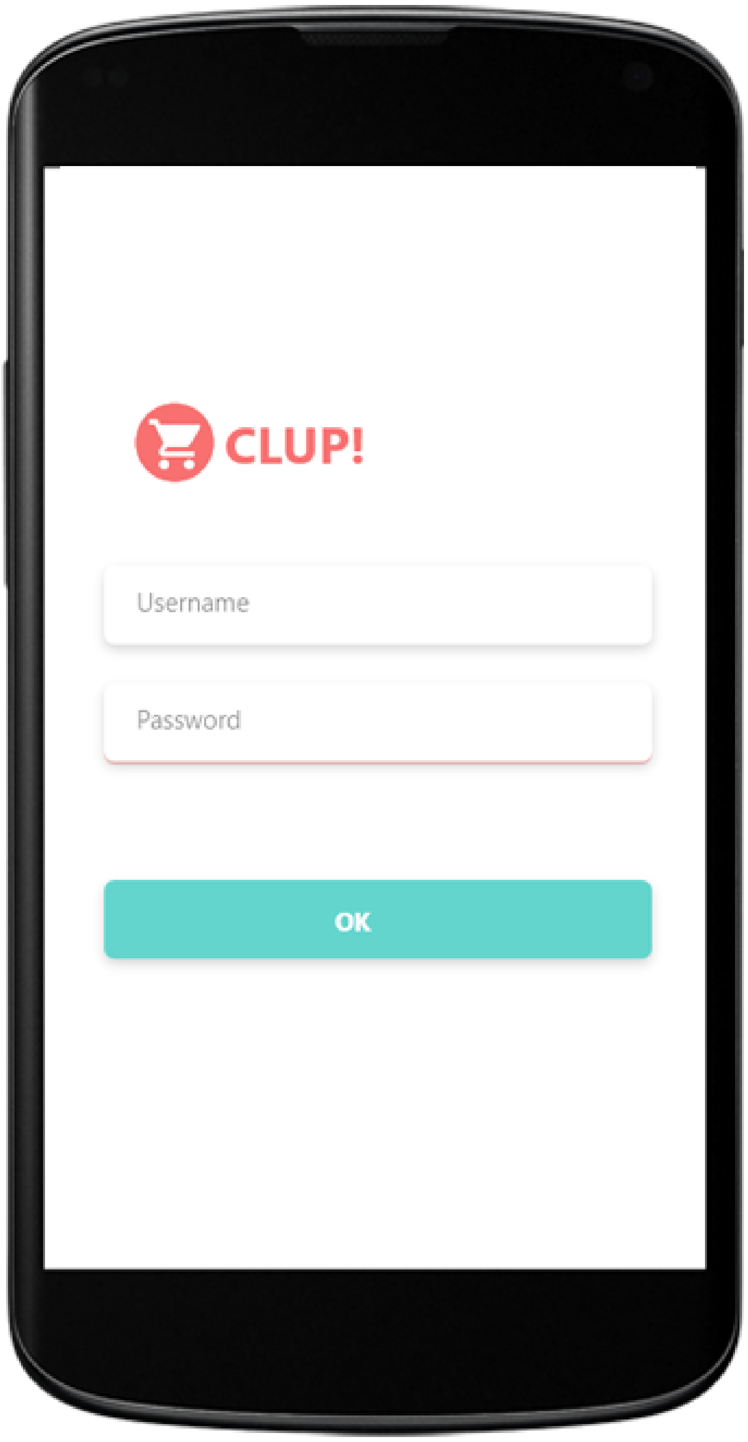
3.1.3 Splash Screen



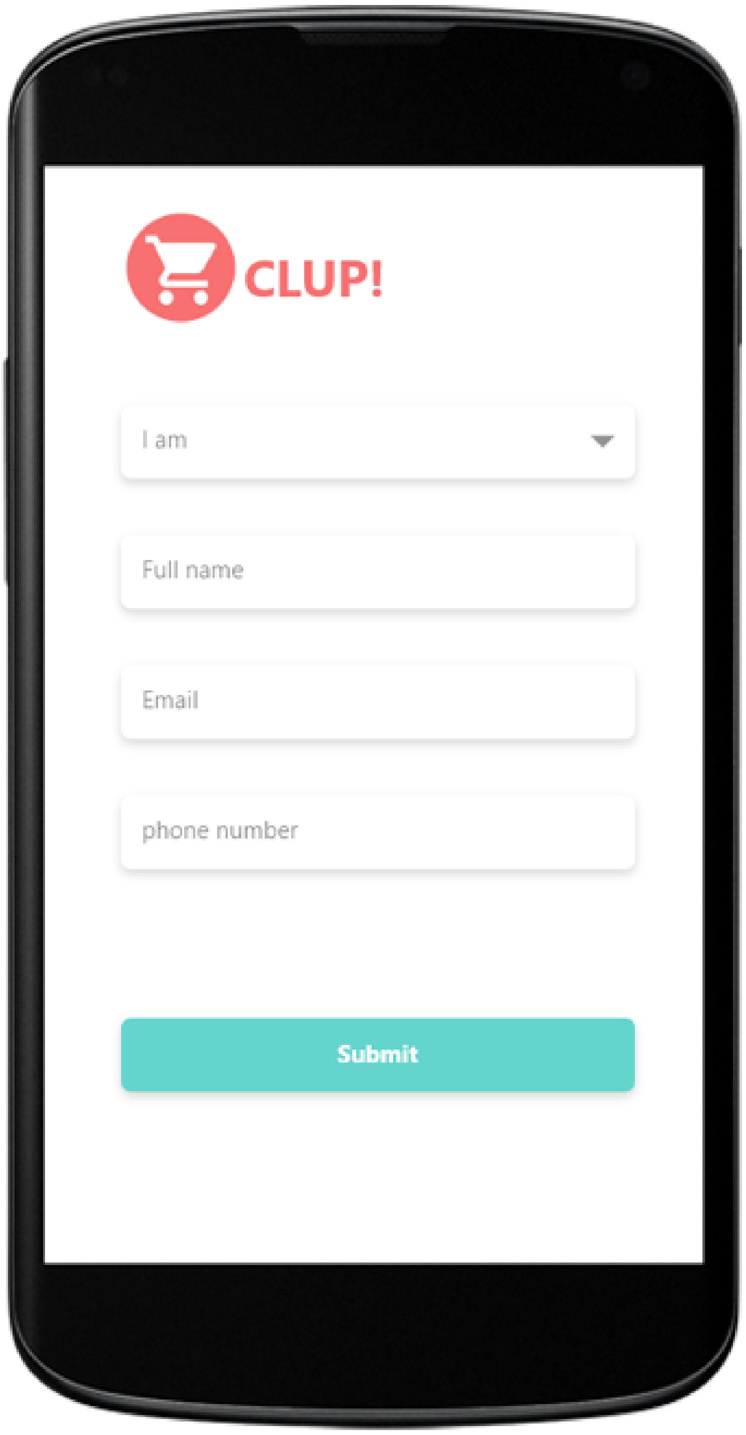
3.1.4 Home



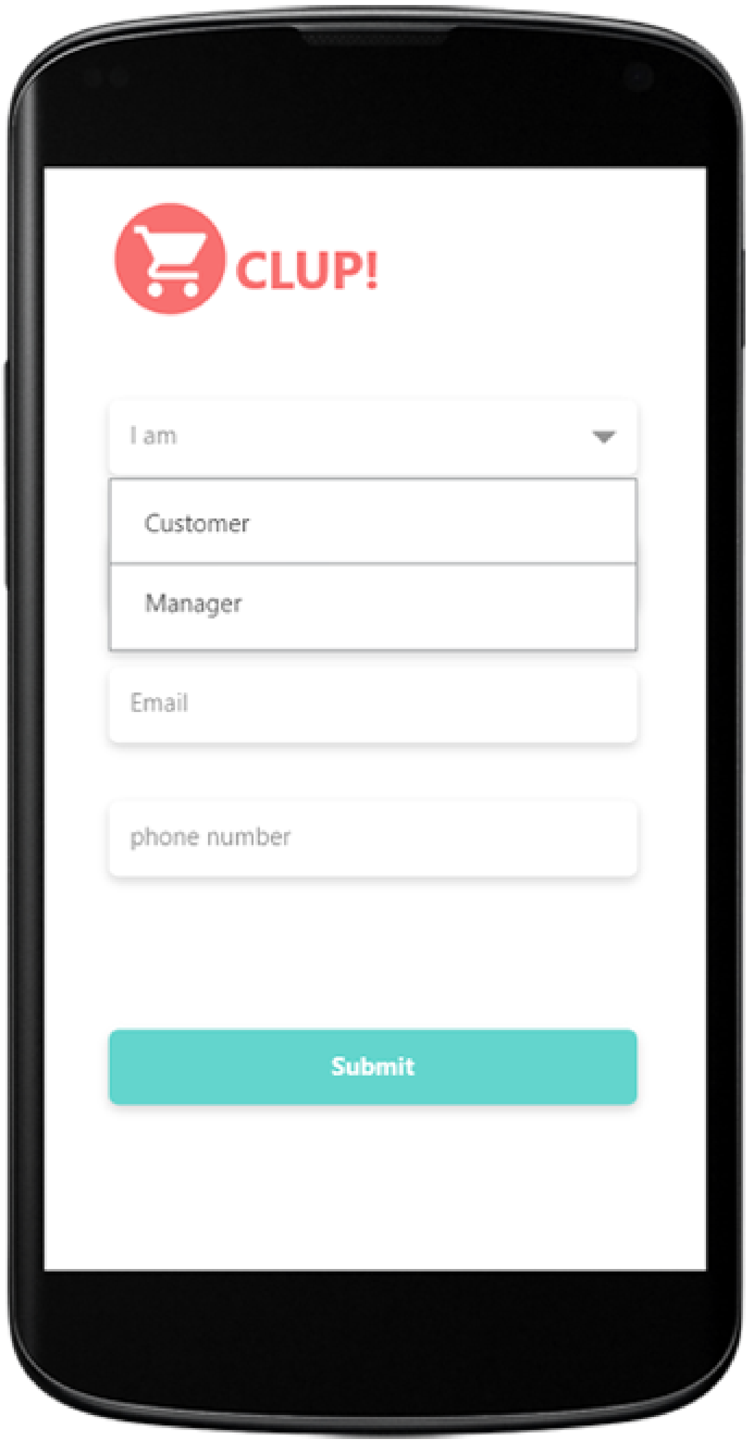
3.1.5 Sign In



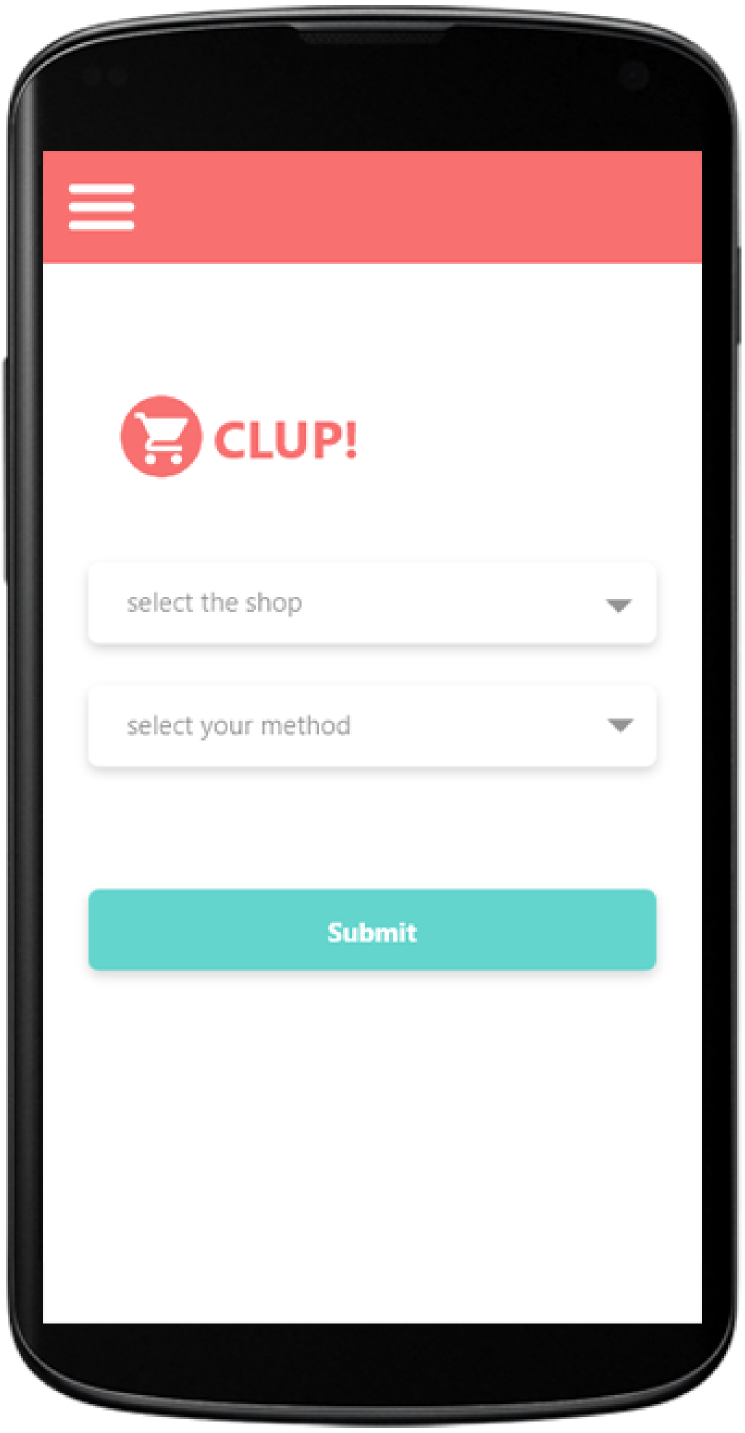
3.1.6 Sign Up



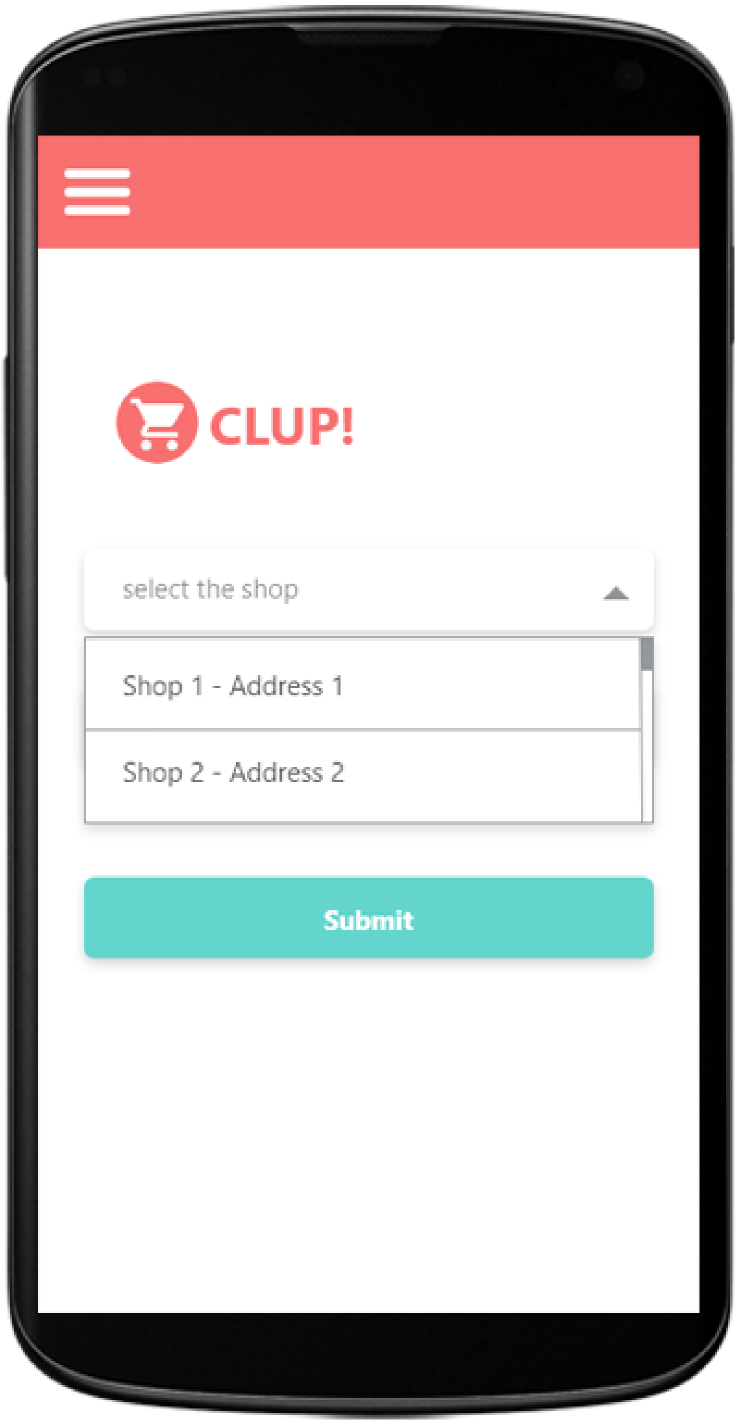
3.1.7 Sign up II



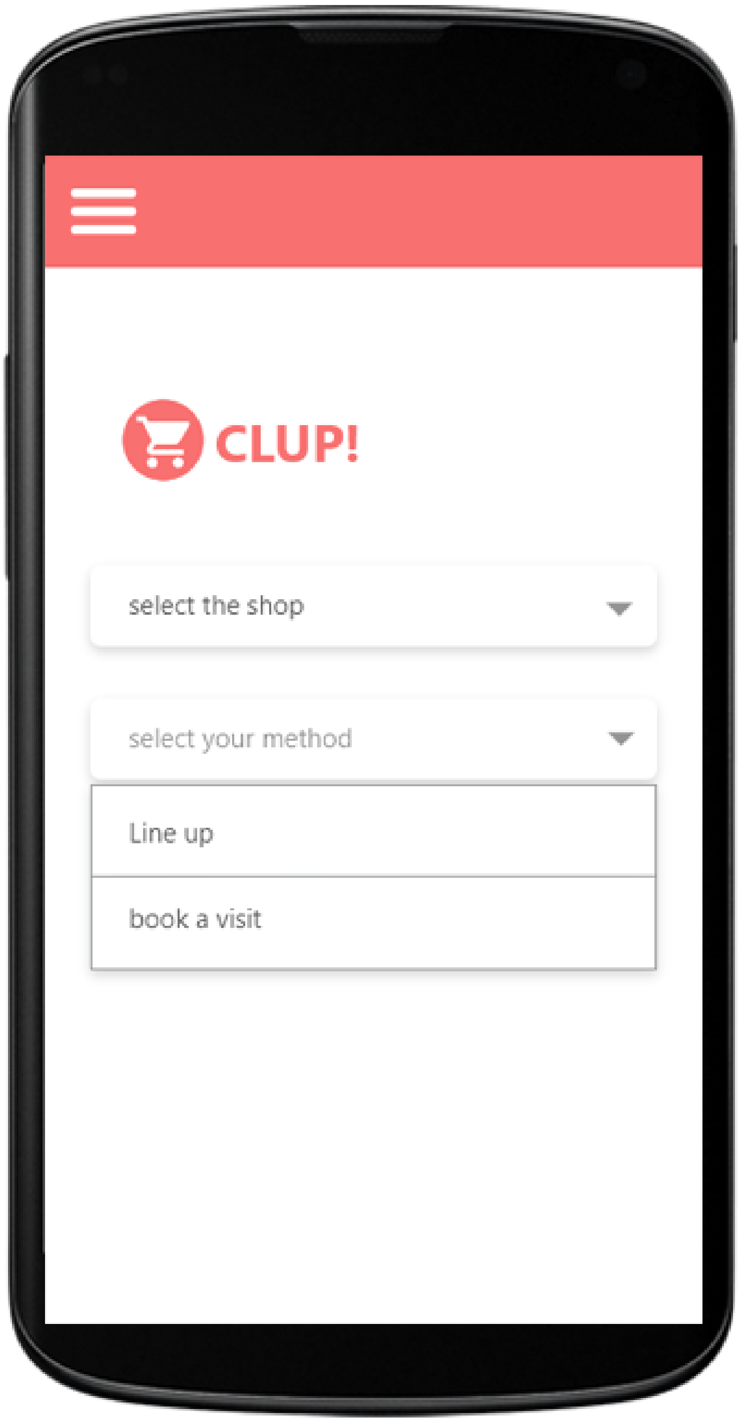
3.1.8 Booking



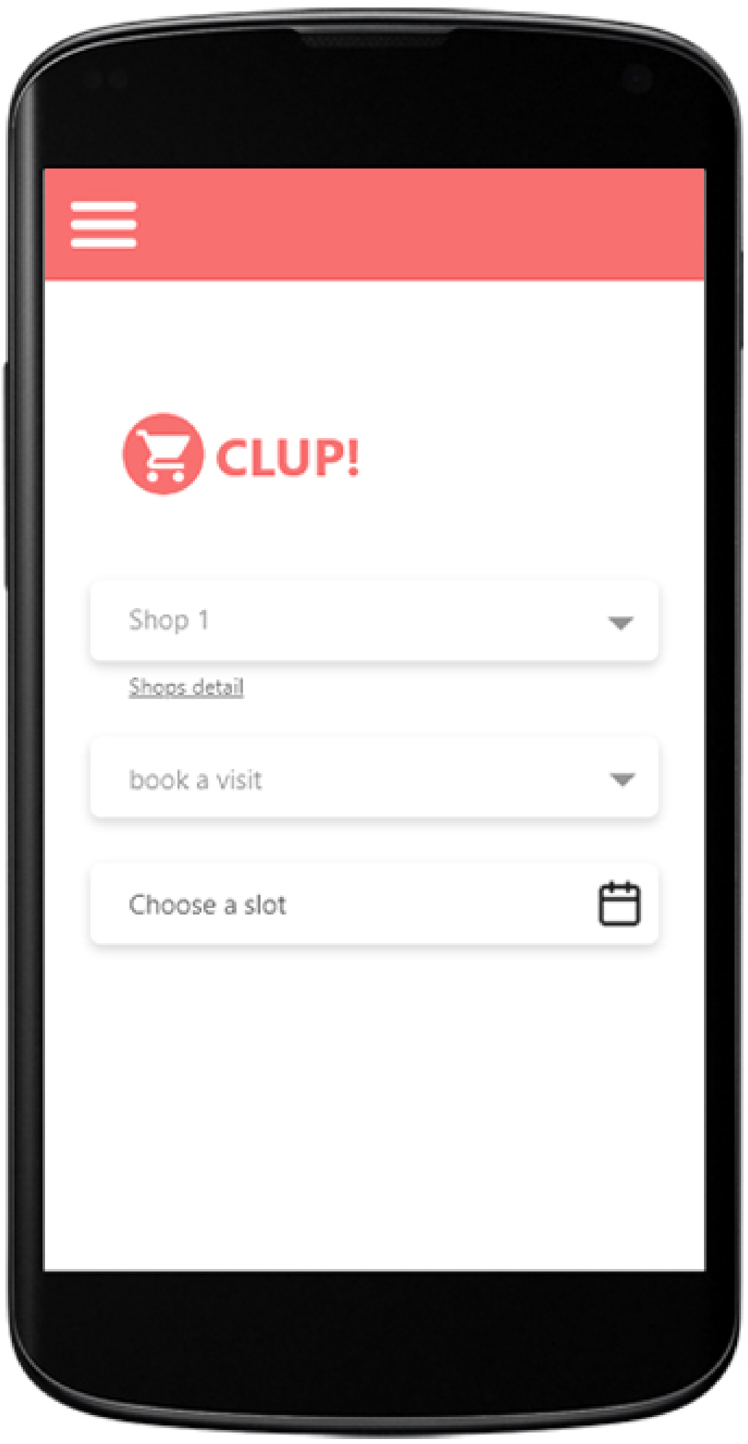
3.1.9 Booking II



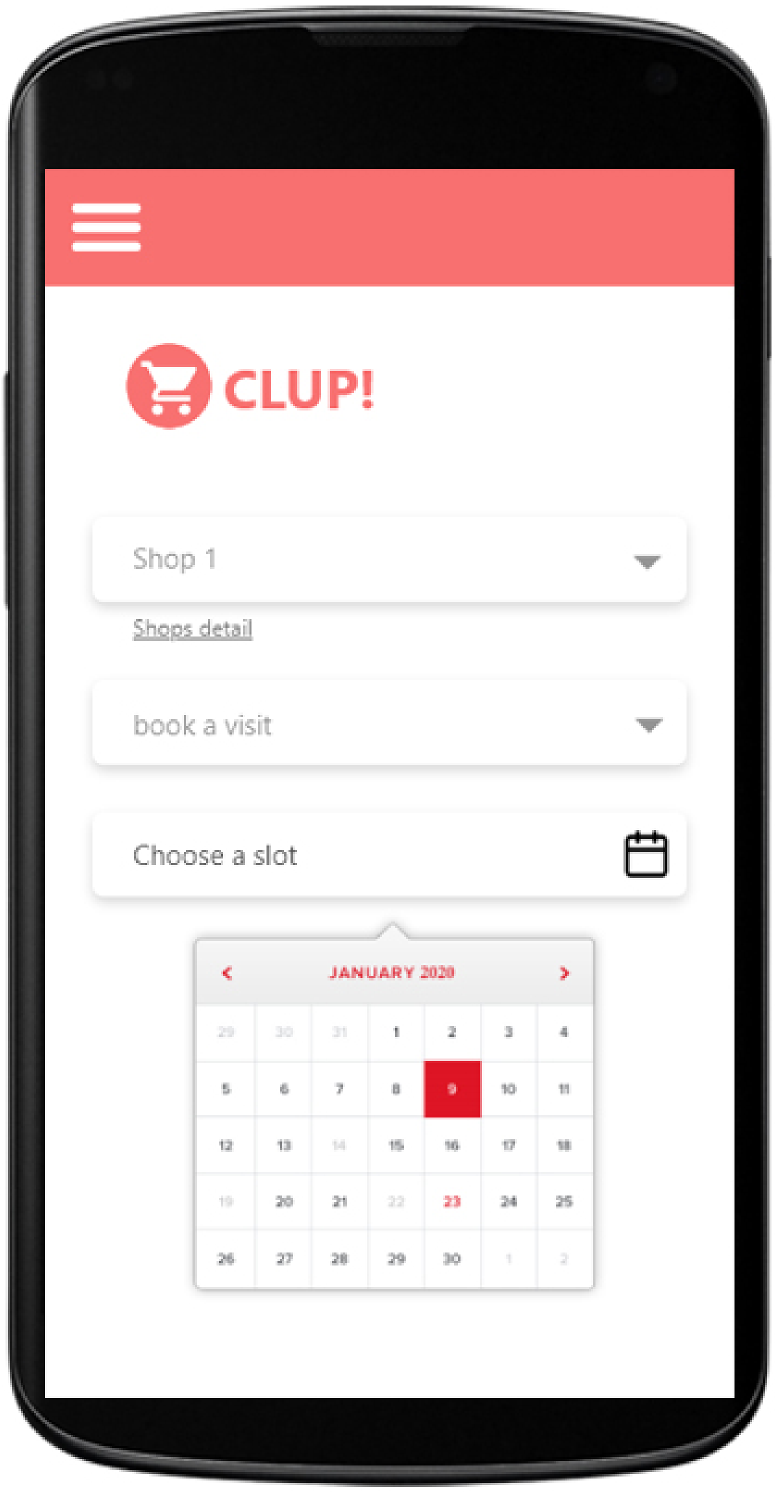
3.1.10 Booking III



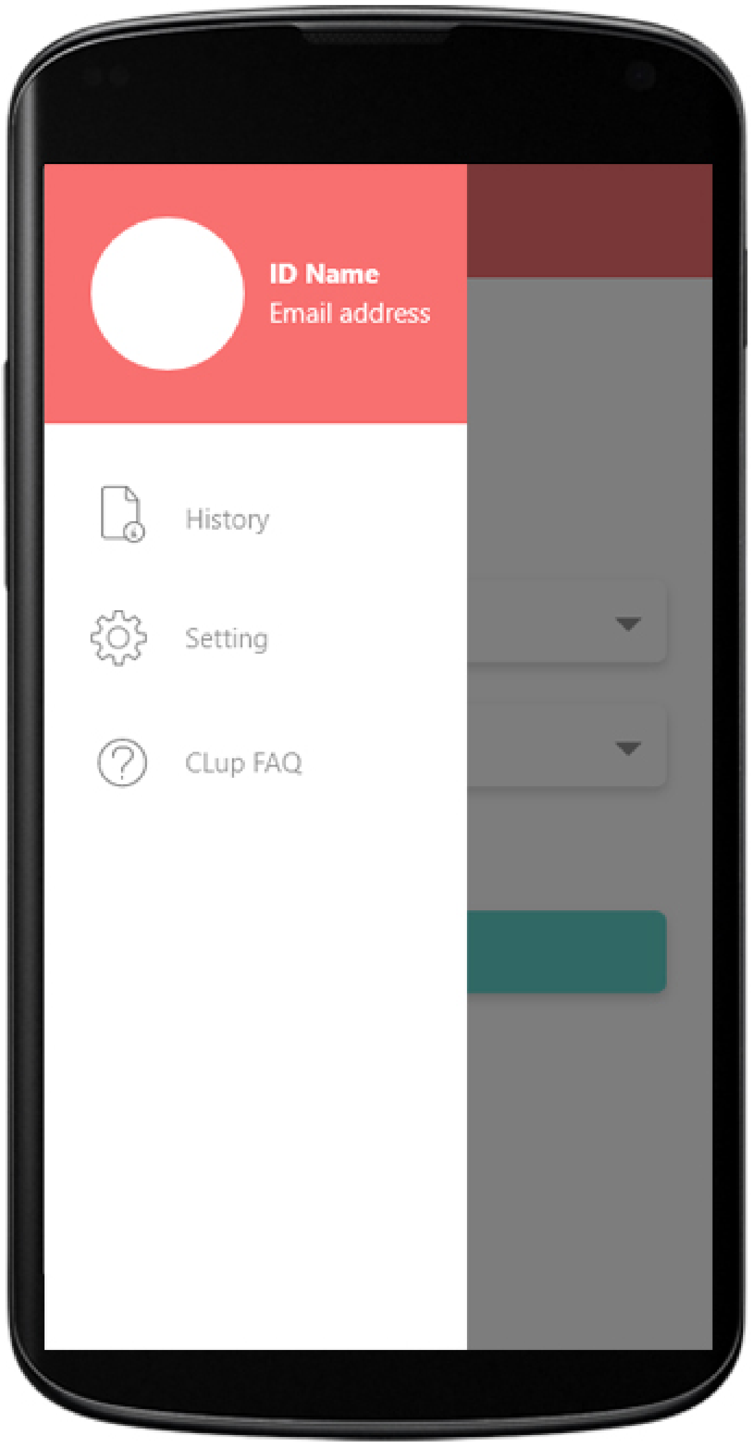
3.1.11 Time reservation for ”Booking a visit”



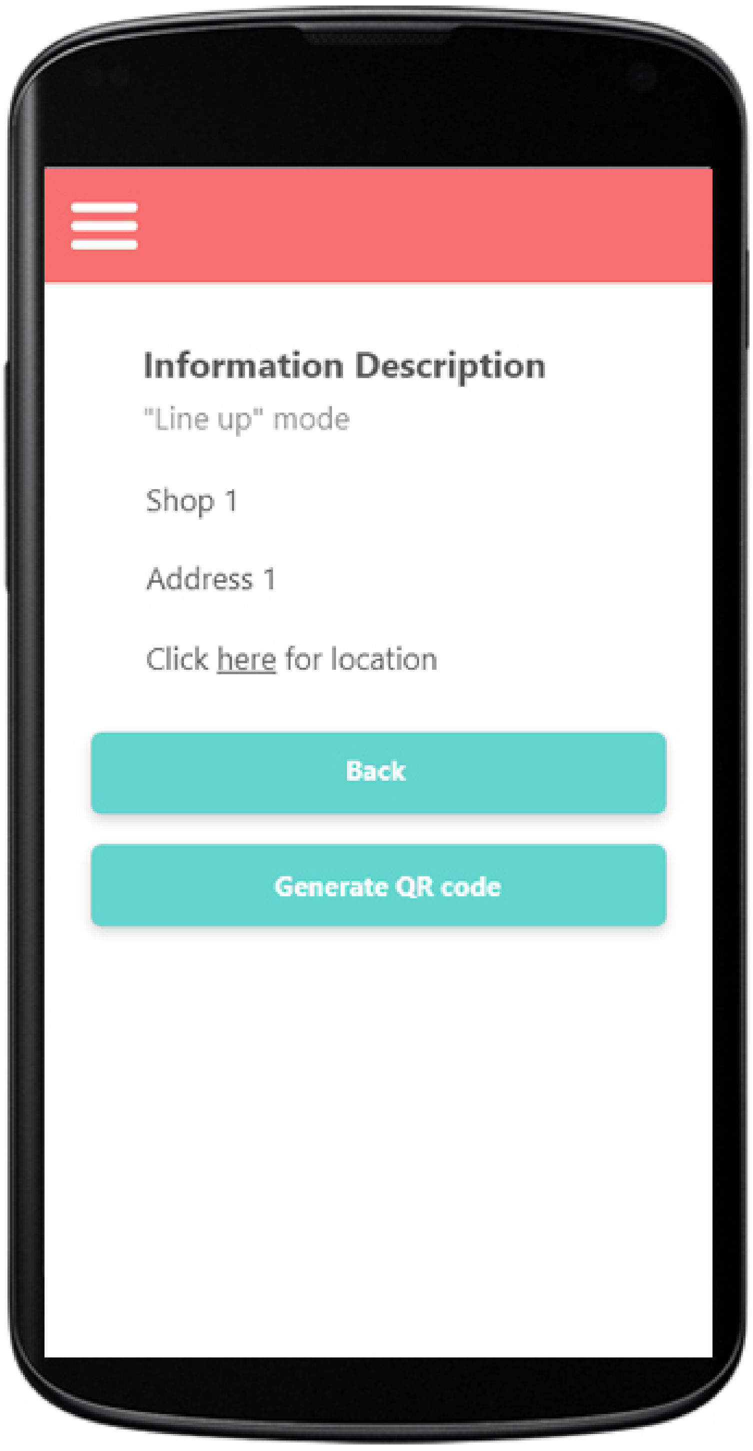
3.1.12 Time reservation for "Booking a visit" II



3.1.13 Hamburger Menu



3.1.14 Information Description(Line-up mode)


Information Description

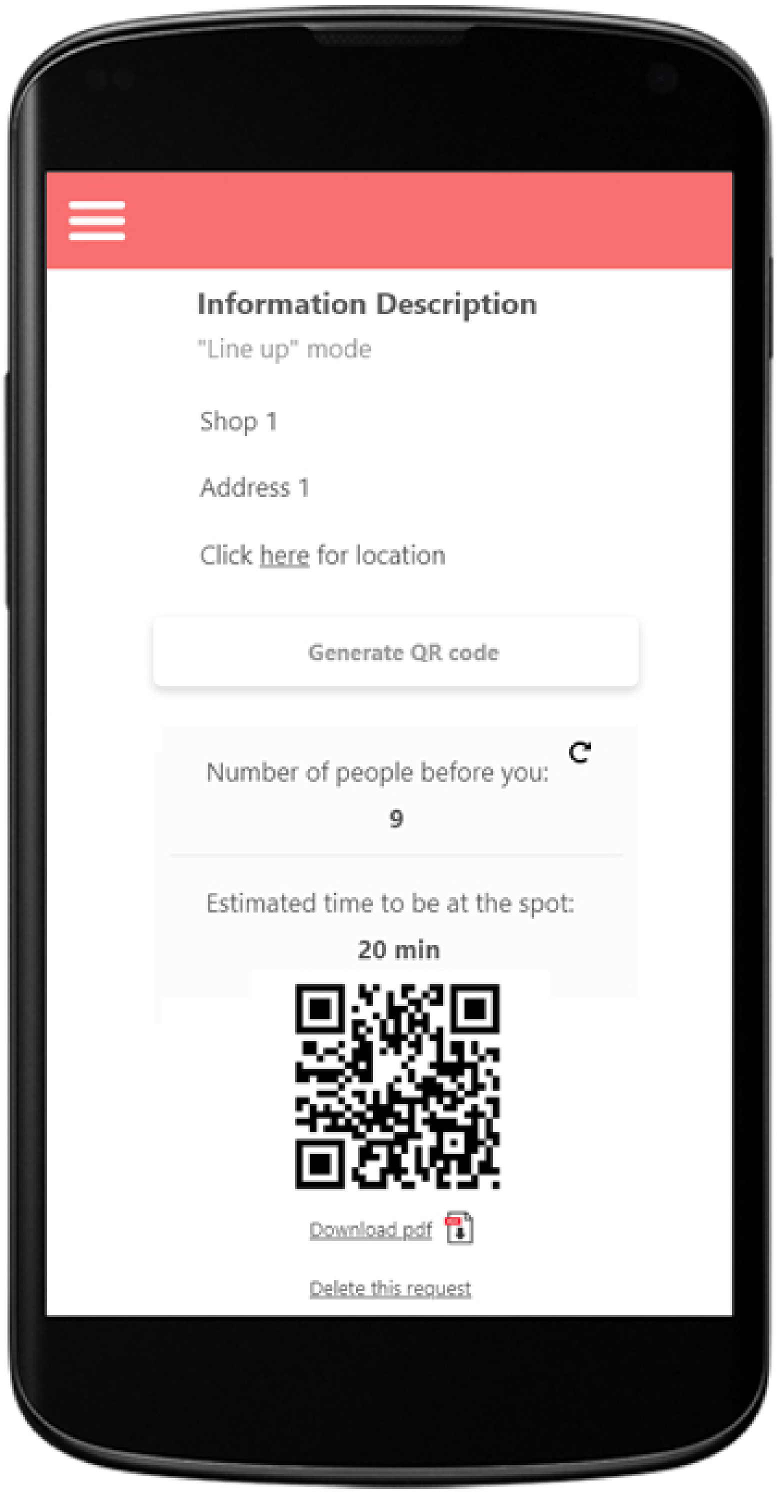
"Line up" mode

Shop 1

Address 1

Click here for location

3.1.15 QR code(Line-up mode)


Information Description

"Line up" mode

Shop 1

Address 1

Click here for location

Generate QR code

Number of people before you:

9

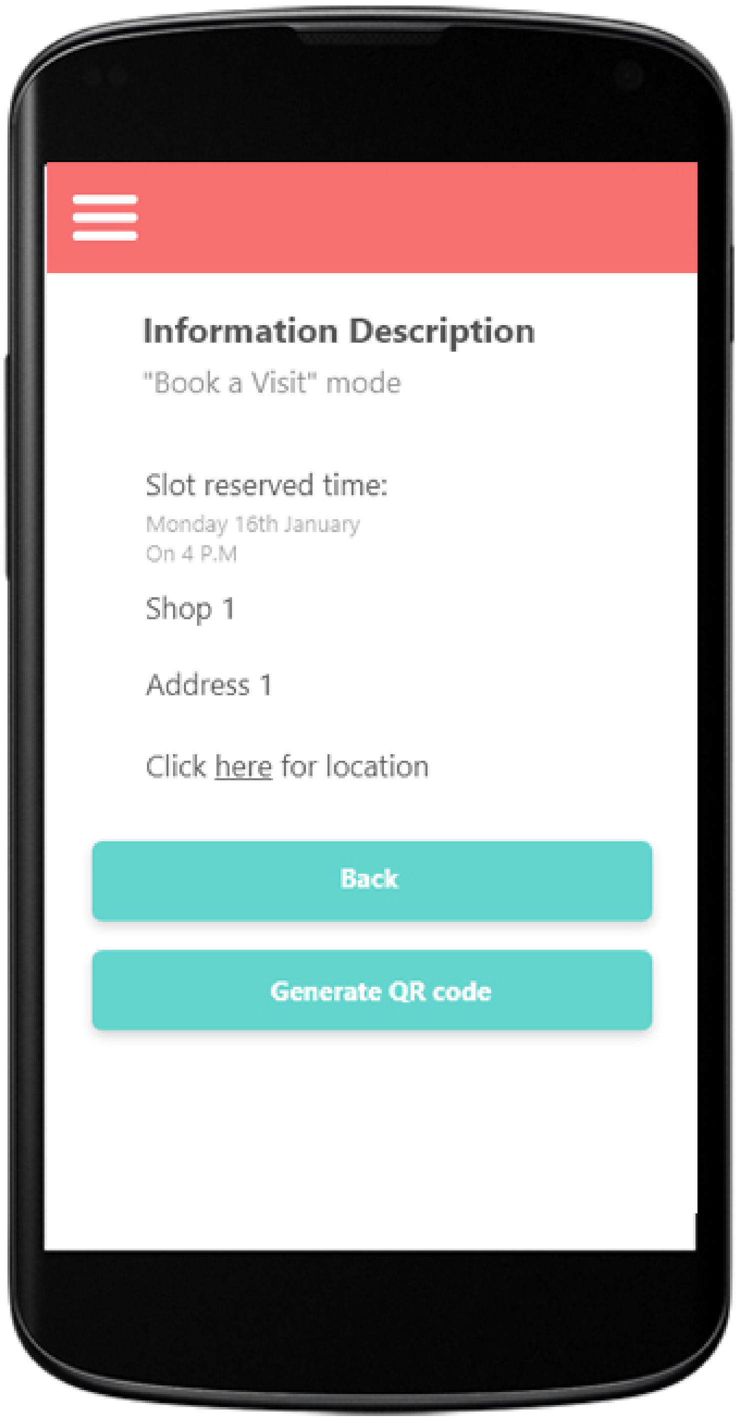
Estimated time to be at the spot:

20 min

Download pdf

Delete this request

3.1.16 Information Description(”Book a visit” mode)


Information Description

"Book a Visit" mode

Slot reserved time:

Monday 16th January

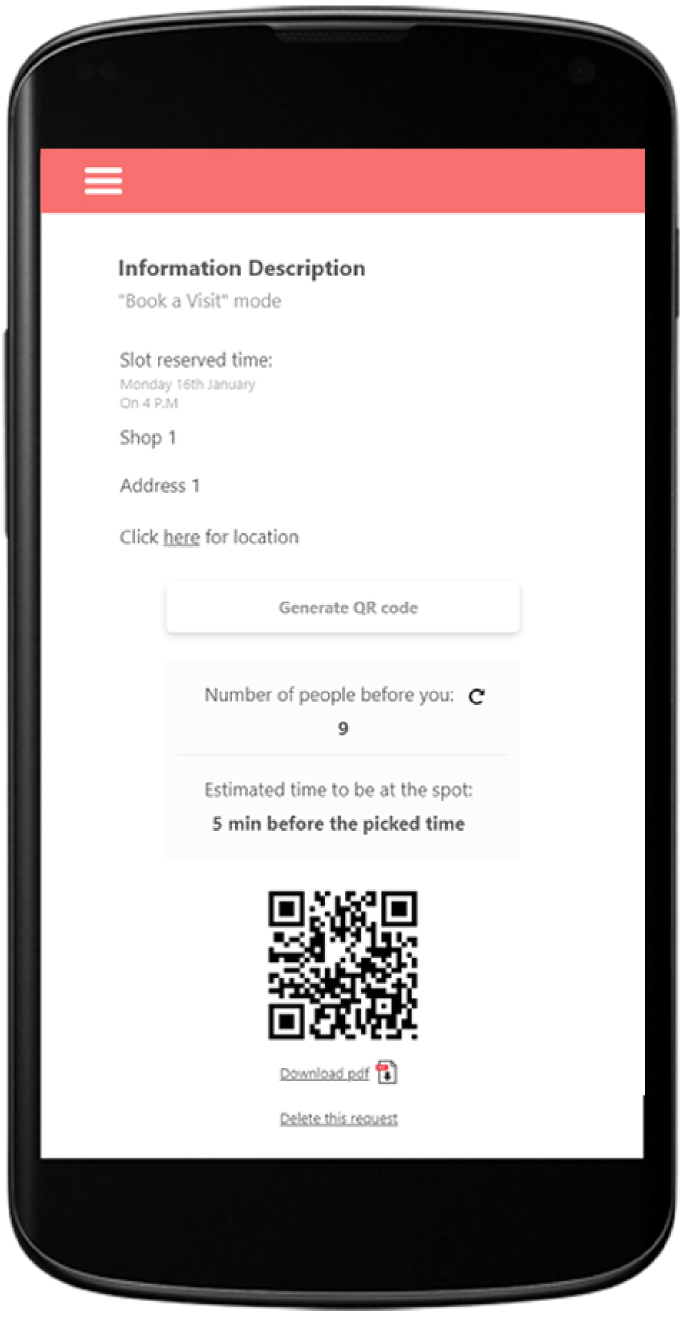
On 4 P.M

Shop 1

Address 1

Click here for location

3.1.17 QR code("Book a visit" mode)



Information Description

"Book a Visit" mode

Slot reserved time:

Monday 16th January

On 4 P.M

Shop 1

Address 1

Click here for location

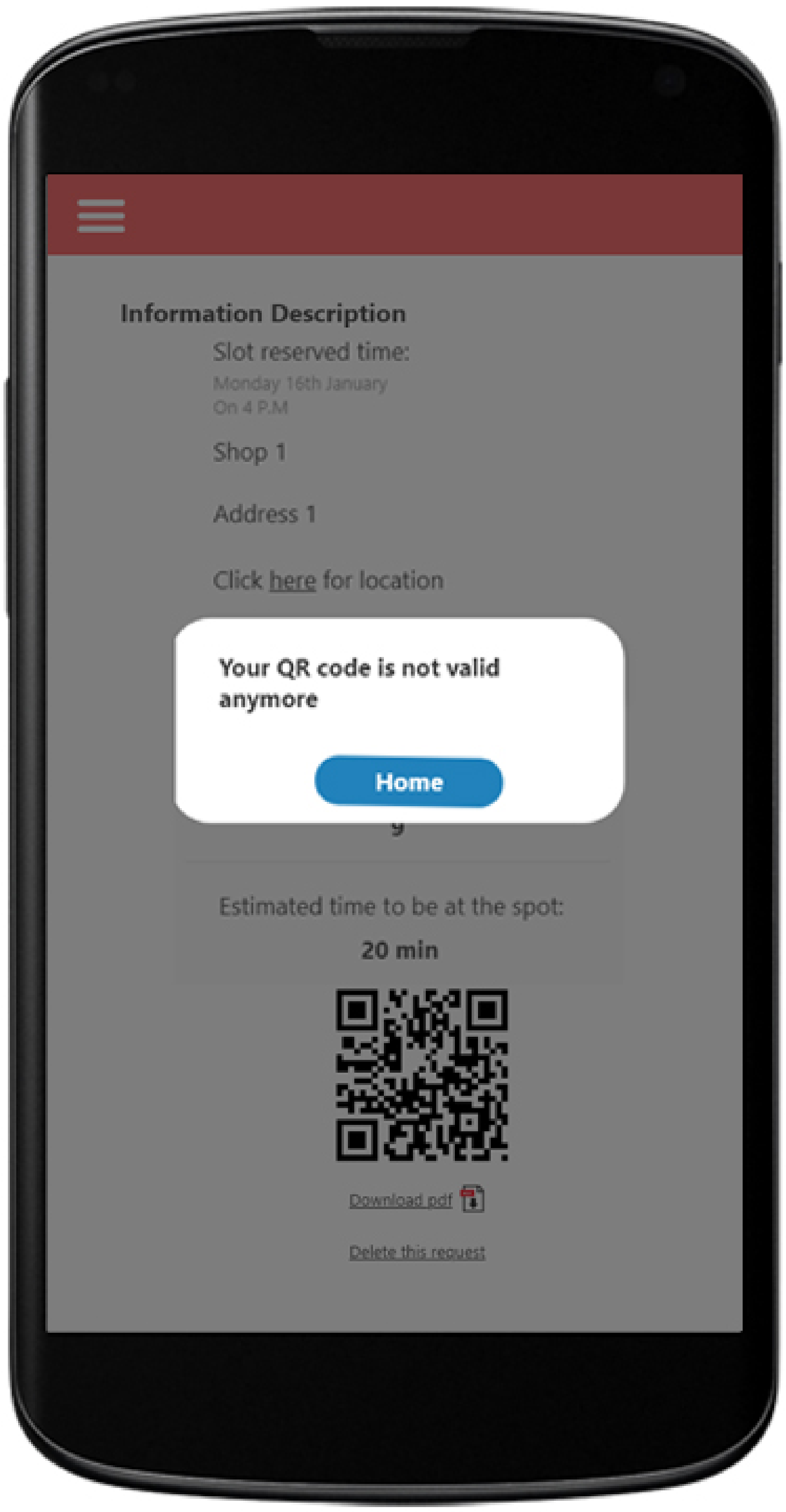
Number of people before you:

9

Estimated time to be at the spot:

5 min before the picked time

3.1.18 Non-valid QR code



Information Description

Slot reserved time:

Monday 16th January

On 4 P.M

Shop 1

Address 1

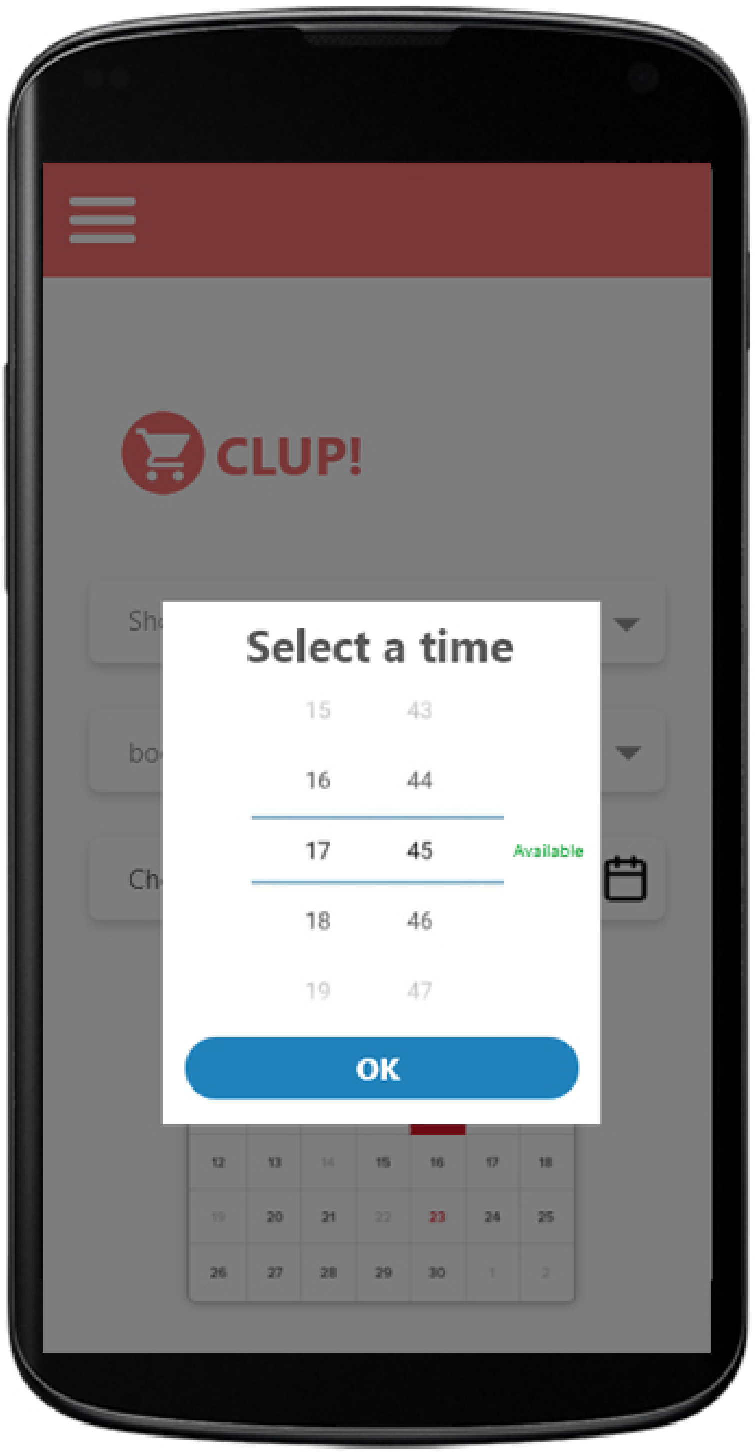
Click here for location

Your QR code is not valid anymore

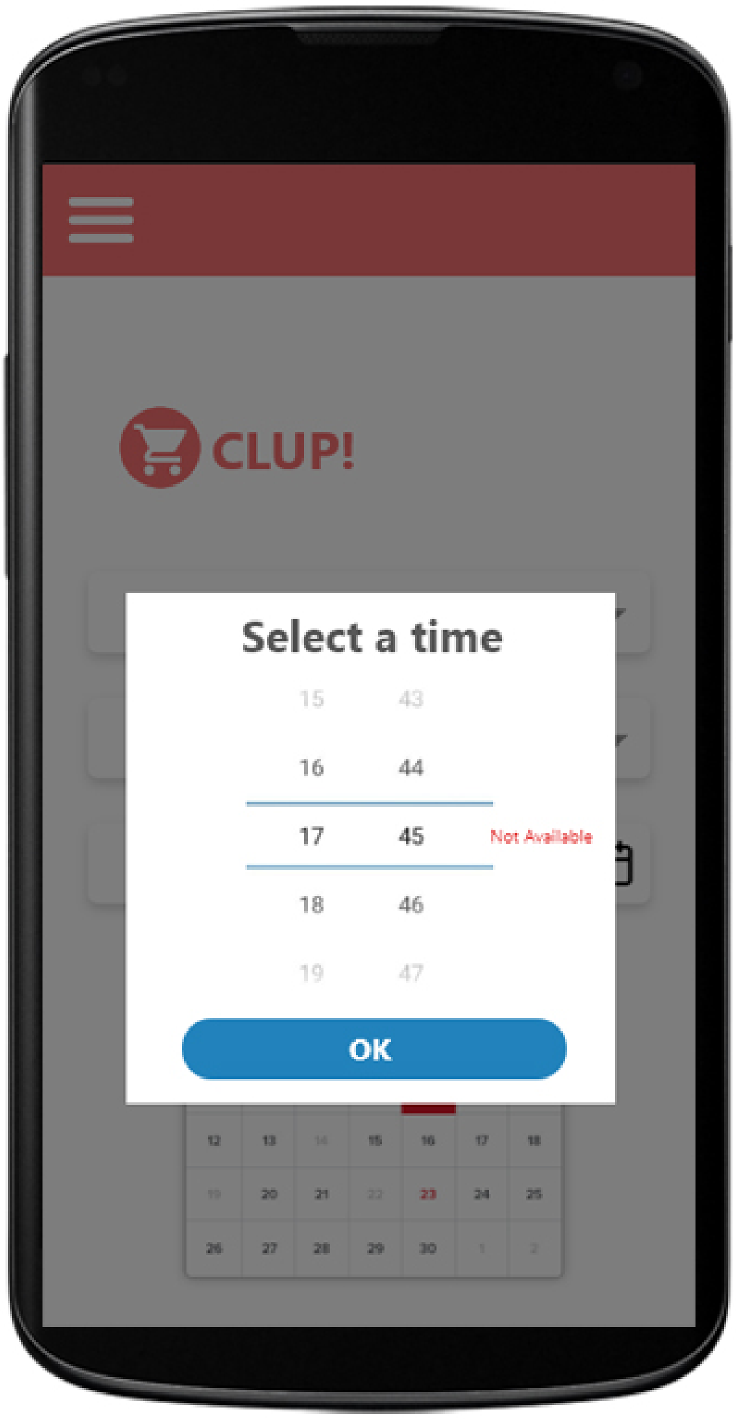
Estimated time to be at the spot:

20 min

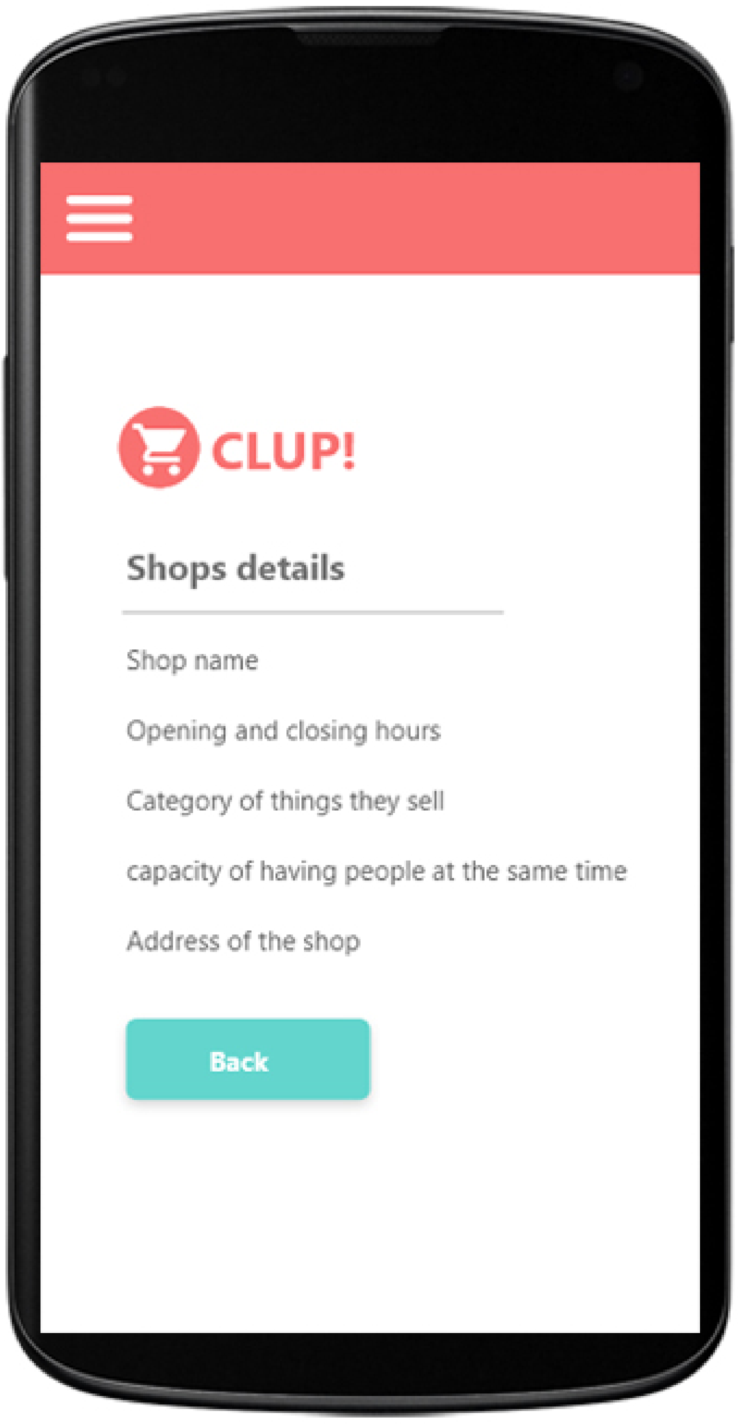
3.1.19 Selecting time for available times



3.1.20 Selecting time (not available)



3.1.21 Shop details



Shop name

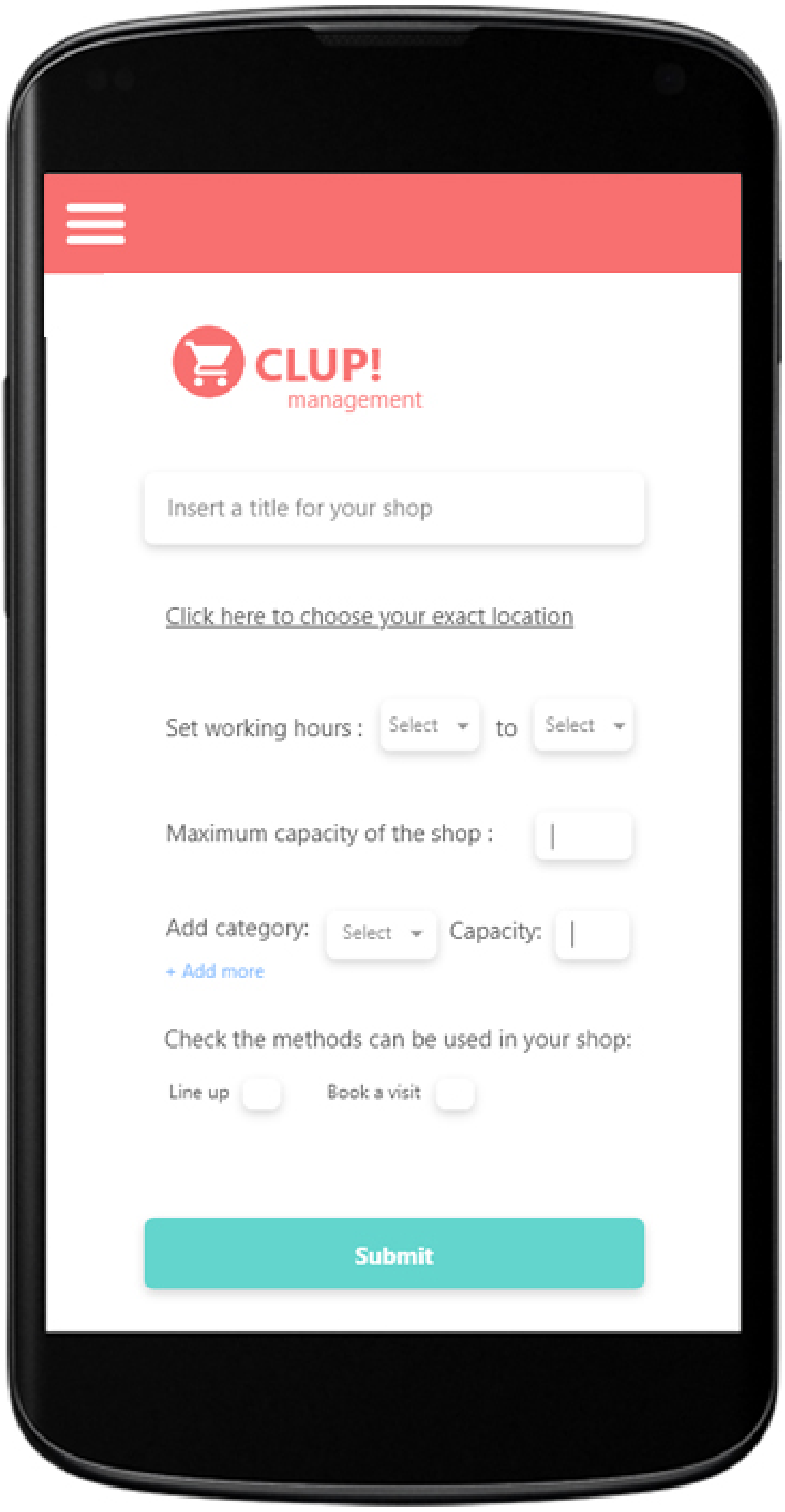
Opening and closing hours

Category of things they sell

capacity of having people at the same time

Address of the shop

3.1.22 Managers Registration Form



Insert a title for your shop

Click here to choose your exact location

Set working hours : Select to Select

Maximum capacity of the shop :

Add category: Select Capacity:

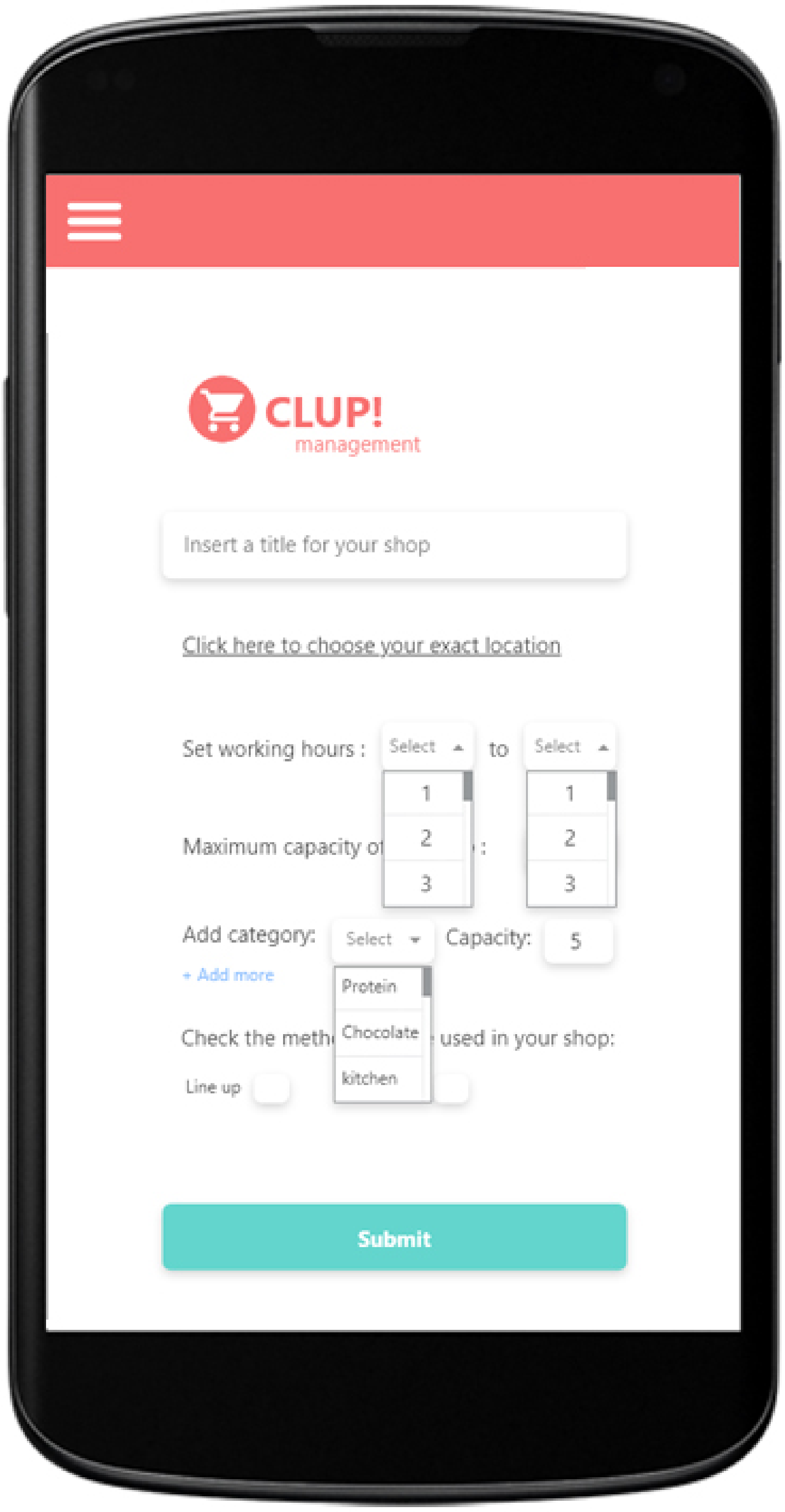
+ Add more

Check the methods can be used in your shop:

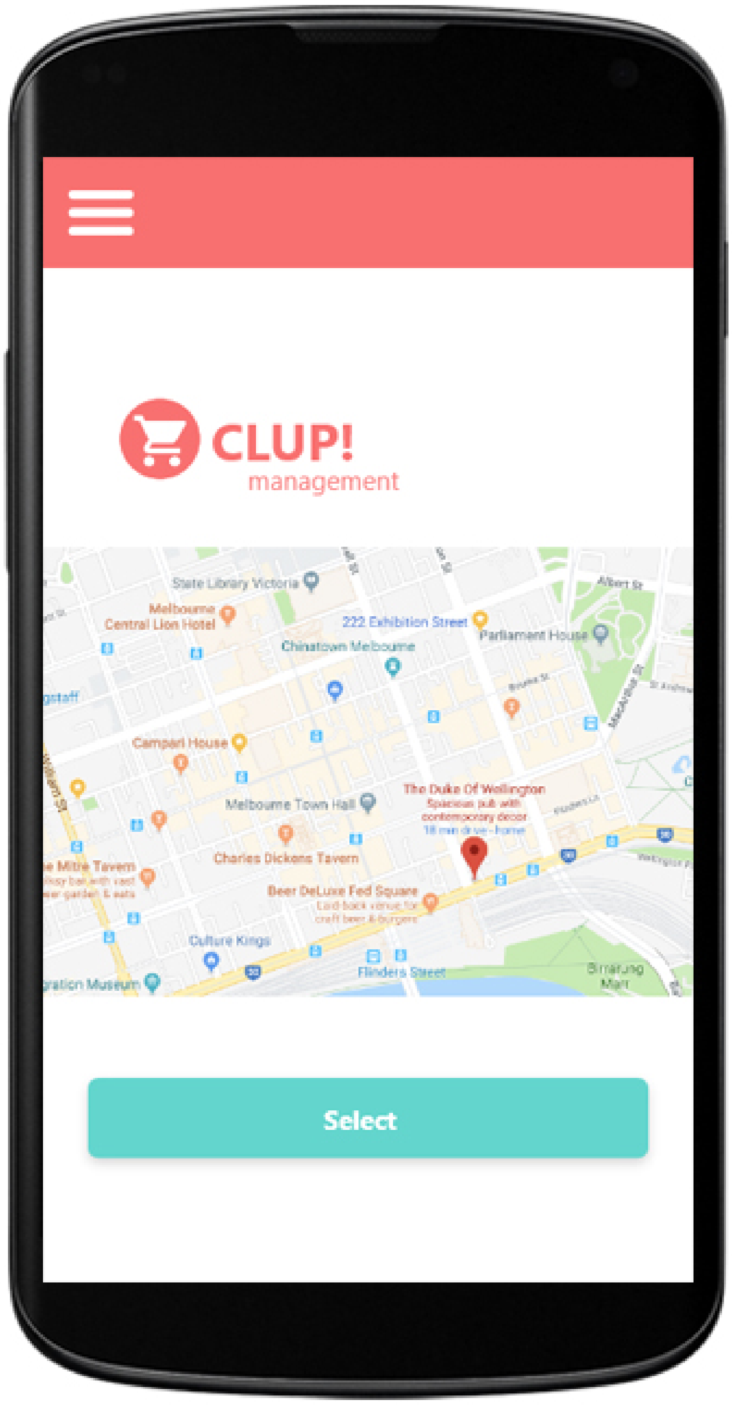
Line up Book a visit

Submit

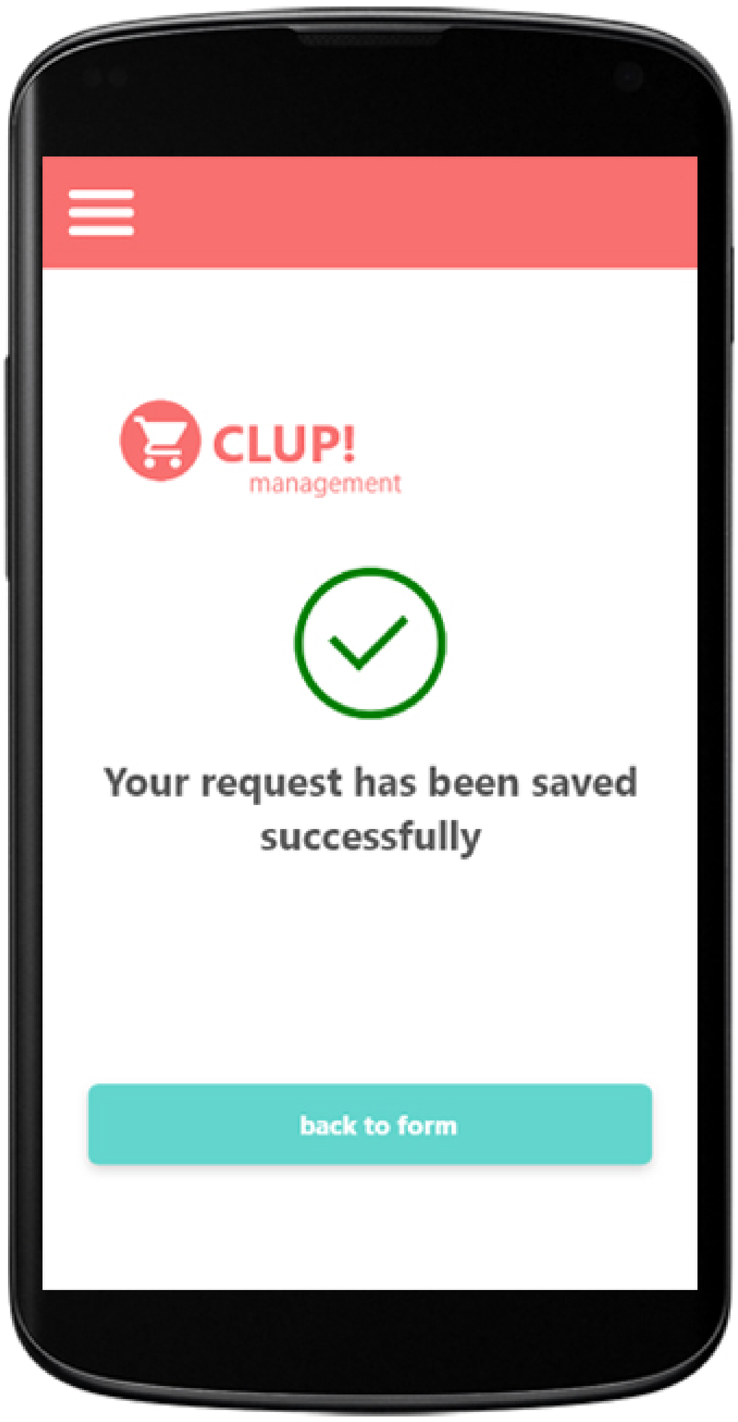
3.1.23 Managers Registration Form details



3.1.24 Selecting Location

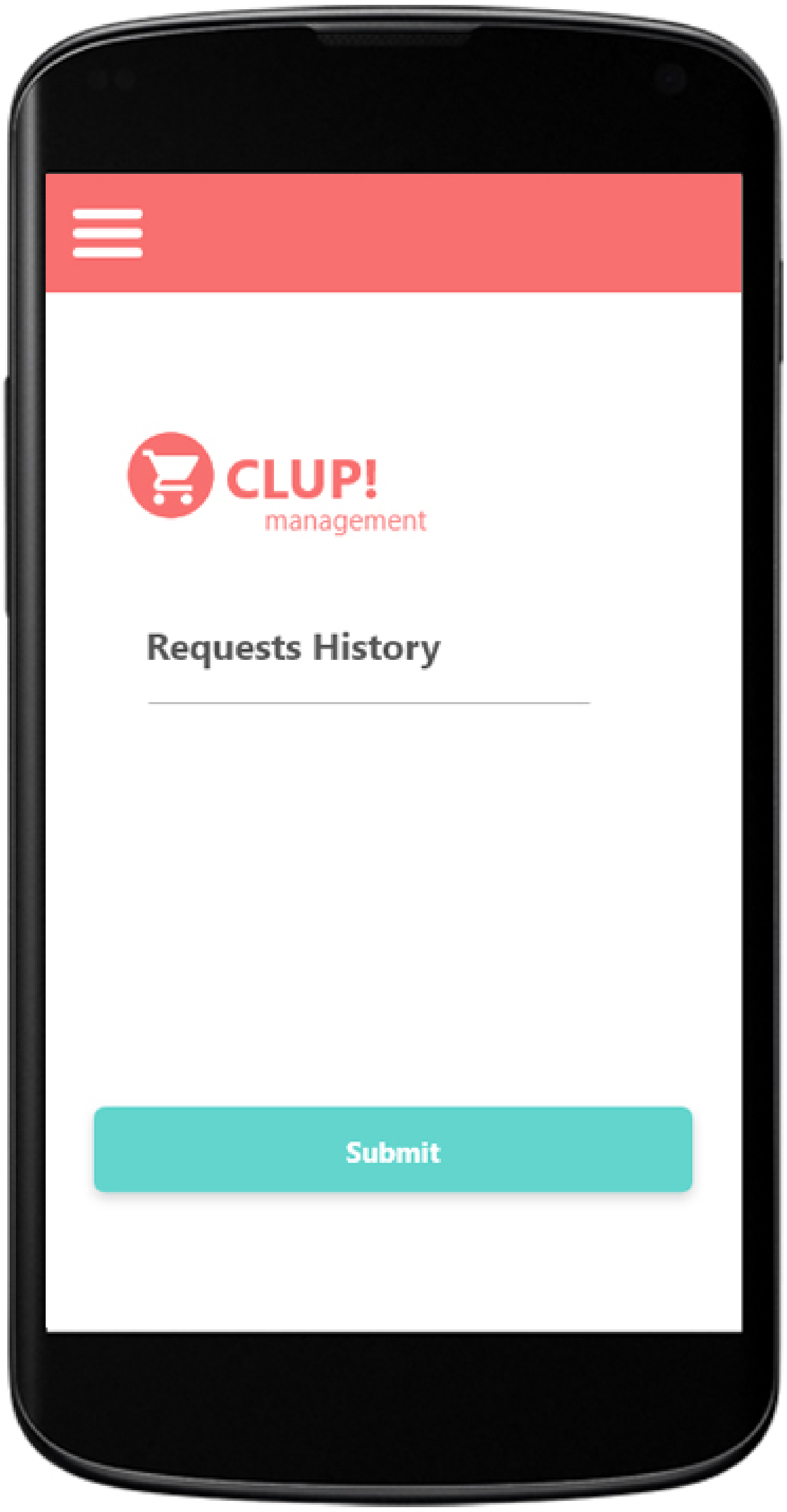


3.1.25 successful request



Your request has been saved successfully

3.1.26 Request history



4 Requirements Traceability

R1	AccessManager DBManager
R2	AccessManager DBManager
R3	AccessManager DBManager
R4	AccessManager DBManager ShopManager
R5	AccessManager DBManager ShopManager
R6	AccessManager DBManager ShopManager
R7	AccessManager DBManager ShopManager
R8	UserMobileApp
R9	UserMobileApp ShopManager MapManager GoogleMap DBManager
R10	UserMobileApp ShopManager MapManager GoogleMap DBManager
R11	TicketHandler DBManager Estimator
R12	BookManager TicketHandler DBManager Estimator
R13	BookManager DBManager Estimator

R14	DBManager TicketHandler
R15	DBManager Estimator
R16	BookManager DBManager
R17	NotificationManager DBManager Estimator
R18	NotificationManager DBManager TicketHandler
R19	WebApp DBManager TicketHandler
R20	WebApp DBManager TicketHandler
R21	WebApp BookManager TicketHandler DBManager Estimator
R22	WebApp DBManager TicketHandler
R23	DBManager TicketHandler Estimator
R24	DBManager Estimator
R25	DBManager Estimator
R26	BookManager TicketHandler DBManager Estimator
R27	DBManager TicketHandler

R28	ShopManager DBManager Estimator
R29	ShopManager DBManager
R30	ShopManager DBManager
R31	ShopManager DBManager

List of requirements

R1	Users certified with an authentication
R2	Managers certified with an authentication
R3	Managers should register to the application with extra mandatory fields
R4	Only managers can create shop
R5	Only managers can update their shop
R6	Managers can activate advance function 1 (allow users booking) on their shop
R7	Managers can activate advance function 2 (send suggestions) on their shop
R8	Users must accept the GPS location for the application
R9	Users could see the list of shops are around them
R10	Users could see the list of shops are around them
R11	Users could see their ticket (QR code) and estimated waiting time
R12	Users could "book a time" to visit specific shop
R13	Users could choose for which categories they go shopping when they book a visit
R14	The system must generate suitable QR code
R15	The system must estimate the waiting time for each user
R16	The system must make current line-up queue based on users booking
R17	The system sends a notification to user to approach shop based on current location of user and location of shop
R18	The system sends a notification to user to go to the shop when it's him/her turn
R19	After user ticket scanned for entrance, system change status of ticket to invalidate for entrance
R20	The clerk must scan the user ticket
R21	The clerk could add a person to current line-up queue
R22	The clerk could print ticket
R23	After the user checked out, the system analyze shopping list and duration time and add it to the user's visit history
R24	The system could estimate the category of shopping and duration of shopping for specific user
R25	The system estimate the time based on user characteristic
R26	Users can ask a clerk to give them a ticket

R27	If specific time past from user turn to enter his/her ticket will be invalidate for entrance
R28	If managers update a shop the estimated time and book visits will be recalculate
R29	Manager choose the capacity of shop
R30	Manager choose the category of each section in the shop and their capacity
R31	Manager choose working hour of the shop

5 Implementation, Integration and Test Plan

5.1 Overview

This section will discuss the implementation, integration and testing of the CLup application. The section aims to prevent any setbacks entailing changes on the project once started its development and do multiple tasks twice.

We all make mistakes. Some of those mistakes are not important, but some are expensive, so the phase of verification and validation takes an important role. Thus, we try to find as many as bugs possible until the release day.

5.2 Implementation Plan

First of all, it is important to divide the project into smaller components in order to have more concrete goals that help keep the developers' motivation high and make a straightforward development of the project. Therefore, we will divide the project in different components, starting from the ones that are responsible of the basic features and going on until the end, where we will develop the most specific ones. It is not a new method, because it has been studied during this course and it is referred to as 'bottom-up' strategy. This method will help providing a better integration of the project tier-by-tier and make different tests of the behaviour of the application before it ends.

In the bottom-up approach, the implementation must start from the lower components up to the top because in this approach the implementation is gradual. In the design, there are some components that rely on other components. Thus, it's better first start implementing with the component that other components use it.

The first step could be implementing the DB manager which is the component implementing all methods that allow to access to the Database and perform queries and updates on it.

After that we could implement the access manager because it is impossible to implement any other component without users. then we will implement the shop manager because in our application there are two important object one of them is User and the other is the Shop. If you think in wider view, you see we want to assign some users to some shops.

Two important service we use are Notification manager and Map manager this two components are connected to external components of our system so it's good to implement them at this stage.

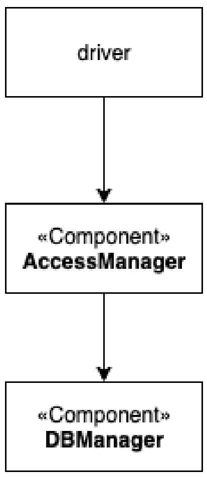
No we could implement the book manager. At this stage, we could slightly see how our application works and books, after we test this part we implement ticket handler so you could see the ticket and generate them. After the ticket handler is good to implement the estimator. estimator is one of the hard components in this application because, it must estimate user behaviors and it must estimate user arrival time and send proper notification to notification manager.

The Redirector and Web server are implemented at the end its role is to dispatch messages coming from the client to the different parts of the system.
The implementation of the UserMobileApp and the TicketMachine, as just said, could be done in parallel with that of the components described before.

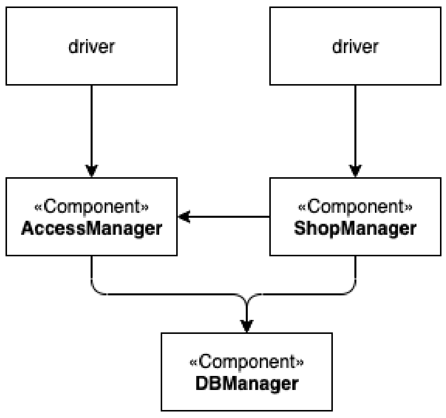
5.3 Integration Strategy

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.

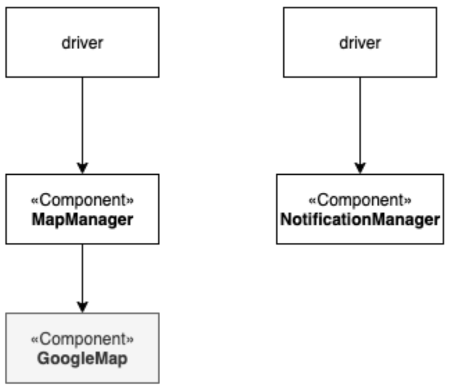
1. At first the AccessManager is implemented and unit tested using a driver for the components that are still under implementation. It is not a complex component, but other components rely on it to provide some services: that's why it is implemented and tested before the others.



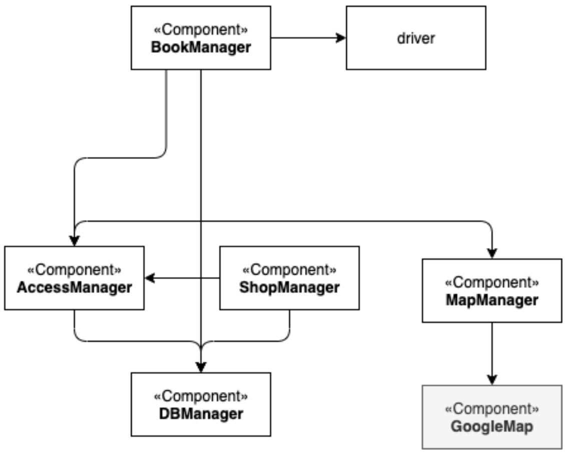
2. Then we implement the ShopManager, because one of the most important enitit y of our application is shop.



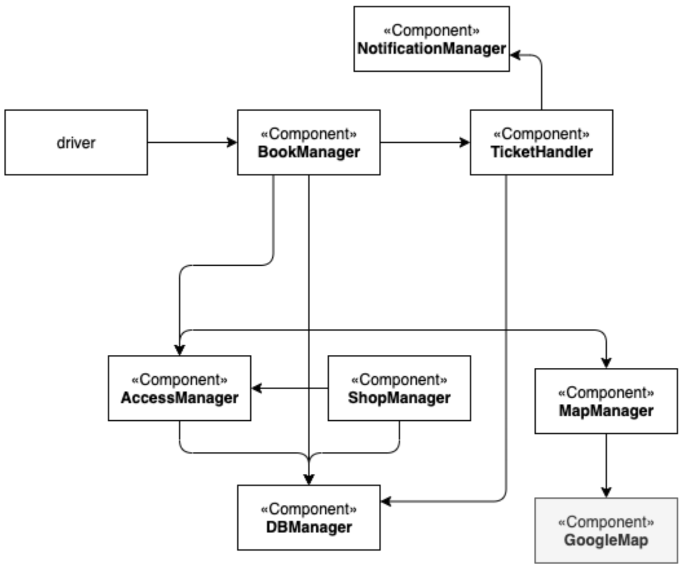
3. Then, to implement the functionality of notifying from the system and receiving it by the user, the NotificationMapManager is needed. However, besides the normal driver, also a stub is needed to cope with the lack of communication with the WebServer that has still to be integrated with the rest. The addition of this stub goes in contradiction with the bottom-up approach but is needed in order to proceed with the integration in the application server.



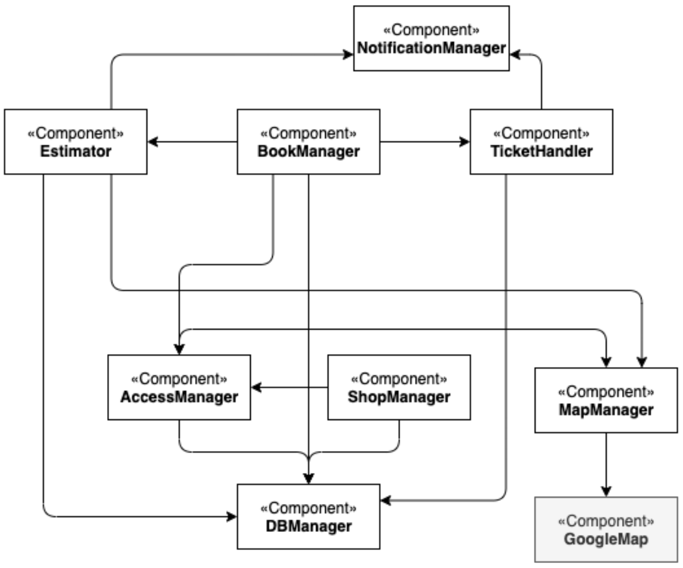
4. to implement the functionality of booking a slot for going to supermarket, the bookingManager is needed. . The addition of this stub goes in contradiction with the bottom-up approach but is needed in order to proceed with the integration in the application server.



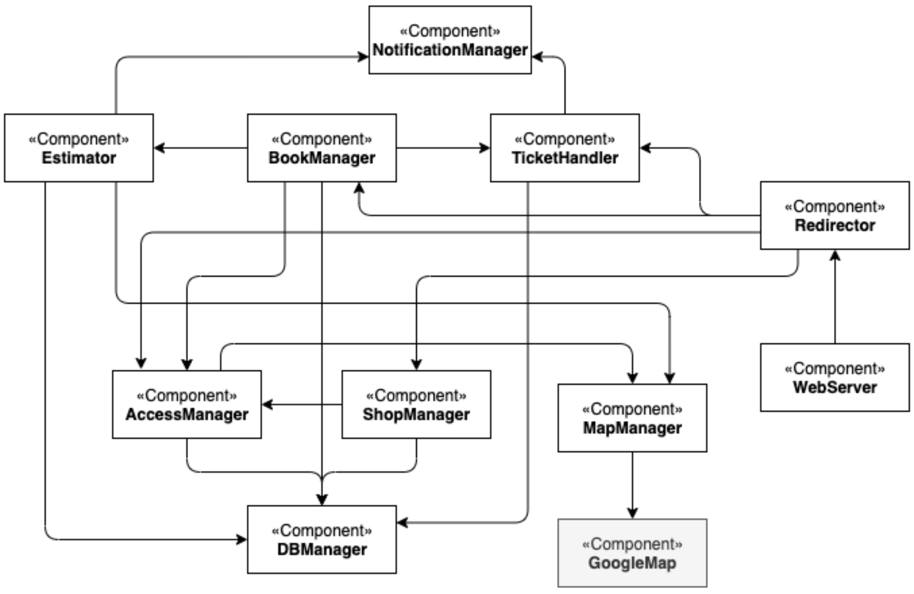
5. at the end of the booking we must generate the ticket, so its good to implement the ticketHandler and integerate it with other part of system.



6. Then the Estimator is implemented, unit-tested and then integrated into the system. It is important to notice that GoogleMapsService will not be unit tested because is offered as a service from a trusted provider: it only needs to be integrated into the system through the Estimator.



7. Then the different drivers are substituted by the Redirector which has to be implemented, unit-tested and integrated with the other components of the application server. its role is to allow the flow of called methods from the client to the server.



6 Effort spent

Saman Fekri:

Topic	**Hours (hh:mm)**
Design meeting	1:30
Introduction	2:00
Component View	4:00
Runtime View	10:00
Reading Deployment technology	2:00
Deployment View	3:00
Study of patterns	3:00
User Interface	2:00
Requirement Traceability	4:00
Implementation Plan	3:00
Integration Strategy	2:00
Revise Sequence Diagram	2:00
Document Review	3:00

Parniya Saeedzadeh:

Topic	**Hours (hh:mm)**
Design meeting	1:30
Introduction	1:30
Component View	2:00
Runtime View	6:00
Reading Deployment technology	2:00
Deployment View	3:00
Study of patterns	3:00
User Interface	8:00
Requirement Traceability	2:00
Implementation Plan	2
Integration Strategy	5:00
Revise Sequence Diagram	1:00
Document Review	3:00

7 References

- All diagrams are draw by the app.diagrams.net (draw.io)
- MVC Definitions derive from Wikipedia
- Three tier Definitions derive from IBM