POLITECNICO
MILANO 1863

M.Sc. Automation and Control Engineering

Software Engineering (for Automation)

Academic year 2021/2022

DESIGN DOCUMENT

CLup: Customers Line-up

Advisor: Prof. Matteo Giovanni Rossi

Students: Alessandro Nocentini alessandro.nocentini@mail.polimi.it

Alberto Valentini alberto.valentini@mail.polimi.it

Fausto Luca Pichierri faustoluca.pichierri@mail.polimi.it

Table of Contents

1. INTRODUCTION

1.1 Purpose

With the Design Document (DD), a more technical overview of the system is given.
The purpose of the document is to provide a functional description of the main architectural components with their interactions and interfaces.
This document is intended to be a support for eventual developers that want to implement a mobile phone application called Customer Line-up (CLup), thought to manage the influx of people inside a supermarket.
It provides a guide during the development process.

1.2 Scope

The coronavirus emergency has put a strain on society on many levels, due to many countries imposing lockdowns that allow people to exit their homes only for essential needs, and enforcing strict rules even when people are justified in going out (such as limiting the number of accesses to buildings and keeping a distance of at least one meter between people).
In particular, grocery shopping a most essential need can become a challenge in the presence of such strict rules. Indeed, supermarkets need to restrict access to their stores to avoid having crowds inside, which typically results in long lines forming outside, which are themselves a source of hazards. In these trying times, people turn to technology, and in particular to software applications,to help navigate the challenges created by the imposed restrictions.
The goal of this project is to develop an easy-to-use application that, on the one side, allows store managers to regulate the influx of people in the building and, on the other side, saves people from having to line up and stand outside of stores for hours on end.
Note: CLup covers only the user experience side.

1.2.1 Goals

- **G1**: Allow Store Managers to regulate the influx of the people in their store
- **G2**: Avoid the creation of hazard gatherings outside the stores and crowds inside it.
- **G3**: Allow customers to save time booking their tickets from home

1.3 Glossary

1.3.1 Definitions

- Customer: person that wants to buy something at the supermarket.
- User: customer with a smartphone that has downloaded the application and uses it.
- Non-User: customer that does not use the application.
- Supermarket/Market/Store: physical shop in which customers buy groceries.
- Store manager: person in charge who administrates the store.
- Store capacity: maximum number of customers allowed in the store at the same time.

- QR Code: type of matrix barcode machine-readable.
- System: sum of hardware and software units dedicated to provide services and features guaranteed by the application.
- Ticket: Element generated by the system containing the QR Code and info about the reservation.
- User city: the city in which the user looks for a supermarket.
- Time slot: A time window of half of an hour.
- Full time slot: time slot that has reached the maximum number of acceptable reservations set by the store manager.
- Past time slot: time slot that is no more selectable because its time window is over at the user time.
- Free time slot: time slot that is not past or full, so available for reservation.
- Reservation time: time window booked at the supermarket by the user.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- CLup: Customer Line-Up (name of the application).
- QR Code: Quick Response code.
- UserGUI: Graphical User Interface

1.3.3 Abbreviations

- Gn: Goal number n.
- Dn: Domain assumption number n.
- Rn: Requirement number n.

1.4 References

- Project proposal: link to the document

- GitHub repository: project repository on GitHub

2 ARCHITECTURAL DESIGN

2.1 Overview

In this chapter a high-level architectural structure of the system is reported.
Further details about the various components and architectural choices are reported in the next sections.

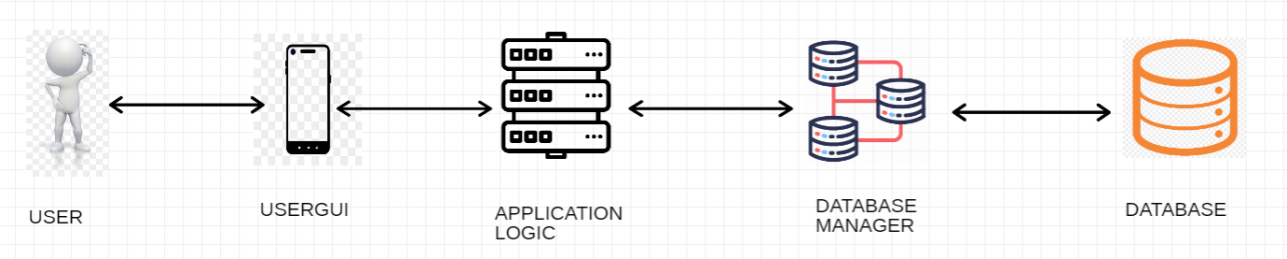


Figure 1

The main components are here reported

- Usergui:
  Thank to the graphical interface of the application installed on the User's phone, he can take advantages of all the offered services.

- Application Logic:
  It is the main component of the system; it elaborates all the request coming from the application. It communicates with the Database manager to receive or store information and with external component like the Mail service.

- Database Manager:
  It reads, writes and updates information, following the instructions given by the Application Logic. It is the only component which communicates with the Database.

- Database:
  It contains all the useful information related to the users and to the markets.

2.2 Component view

In this section, the high-level components previously described are analyzed in terms of their subcomponents, so that, one possible internal modular structure is reported.

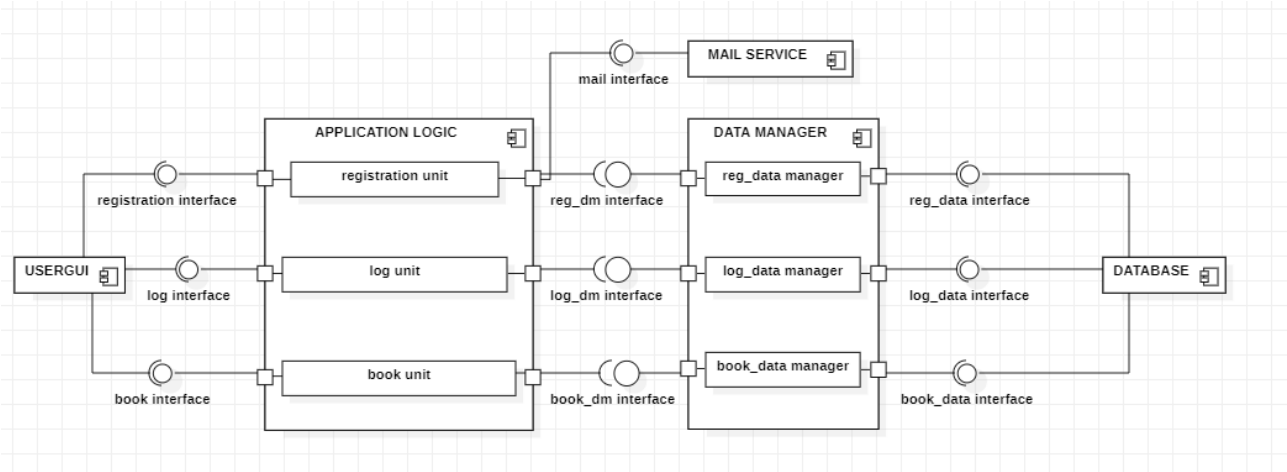


Figure 2

2.2.1 UserGUI

Thanks to this graphical interface the user can take advantages of the CLup-services. Depending on the type of request, the UserGUI communicates with different subcomponents of the system by means of different interfaces.

2.2.2 Application logic

Depending on the type of request coming from the user, the 'Application logic' can be divided in:

- Registration unit:
  It allows a new customer to sign up; it also makes a check on registration form and communicates with the mail system

- Log unit:
  It allows the user to sign in by checking the log in form; it also finds information related to the User's city

- Book unit:
  It allows the user to create a new reservation, to generate the QR code, to cancel some tickets. It is also responsible to show the user the only available time slot and to update the timetable.

2.2.3 Data Manager

The 'Data Manager' is the only component that can get access the Database. It can read or replace the information through different interfaces. It can be divided into:

- Reg_data manager:
  It passes the users information to the 'Registration unit' when someone wants to sign up. It is also able to store modifications.

- Log_data manager:
  It gives to the 'Log unit' information related to the user that wants to sign in, like credentials and default city.
  It is also able to store modifications in the User data.

- Book_data manager:
  It gives to the 'Book unit' all the information that are needed when the logged-in user wants to make a reservation. It is also able to store modifications.

2.2.4 Database

The Database contains all the useful information.
It can be organized into two large blocks:

- Data list:
  It contains all the information and tickets of all the subscribed users.

- Cities list:
  It contains all the information of all the accessible markets.

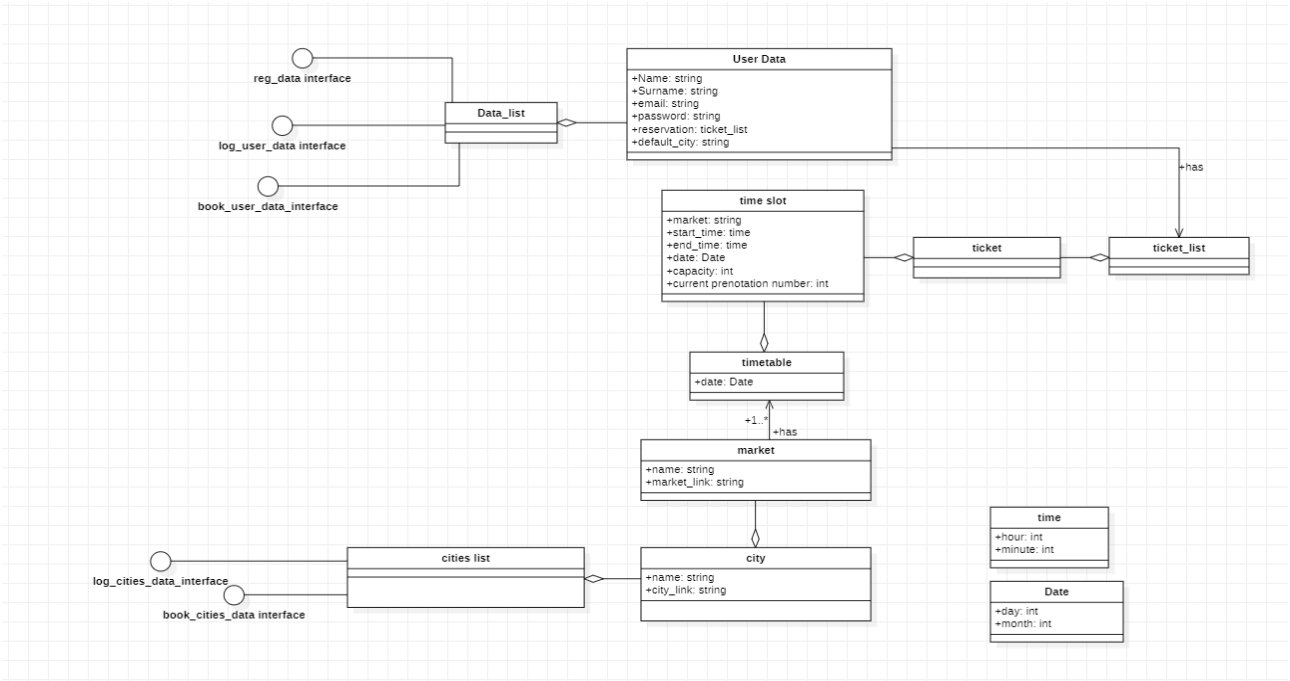


Figure 3

2.3 Component view - extension

Considering a different prospective, the system can be divided in three different separate layers, each one with a different functionality.

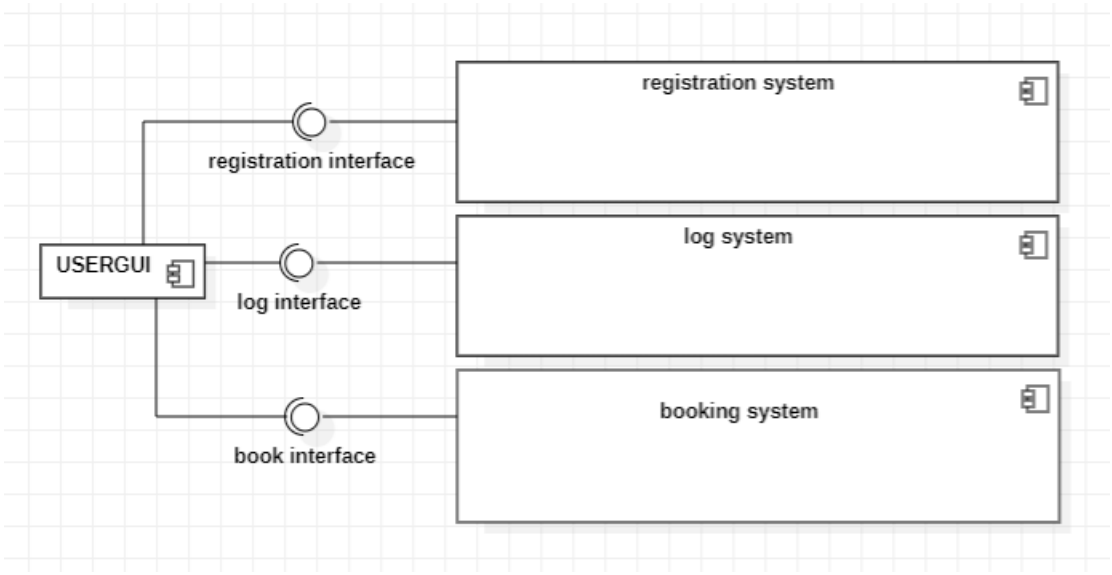


Figure 4

As we can see from figure 4 we can identify different types of subsystems:

- Registration system:
  It must allow a new customer to sign up; this case, it must check if a user is already registered. It should send a confirmation mail.

- Log system:
  Allow the user to sign in by checking the inserted credentials, it also keeps track on the user city.

- Booking system:
  Allow the user to make reservations and to manage its own tickets.

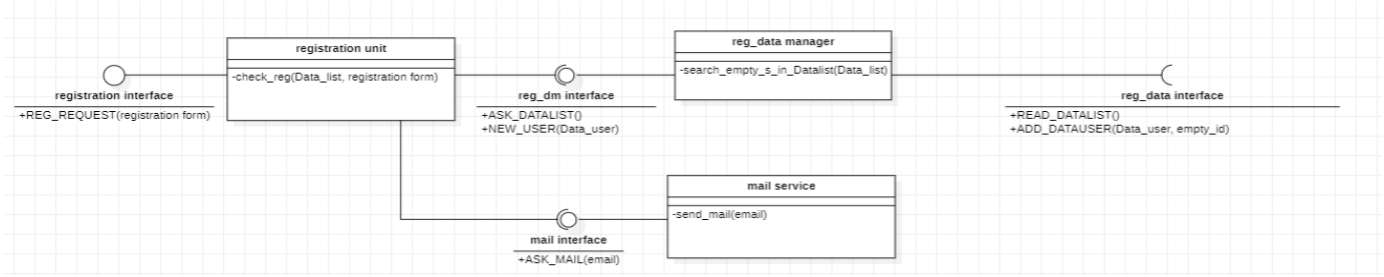


Figure 5 – Registration system

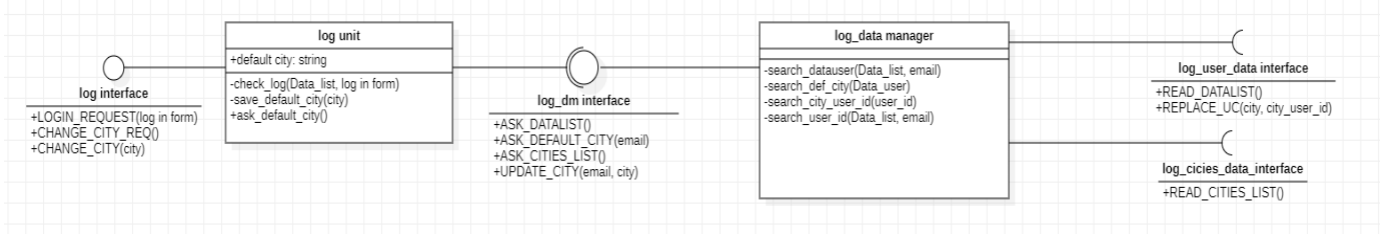


Figure 6- Log system

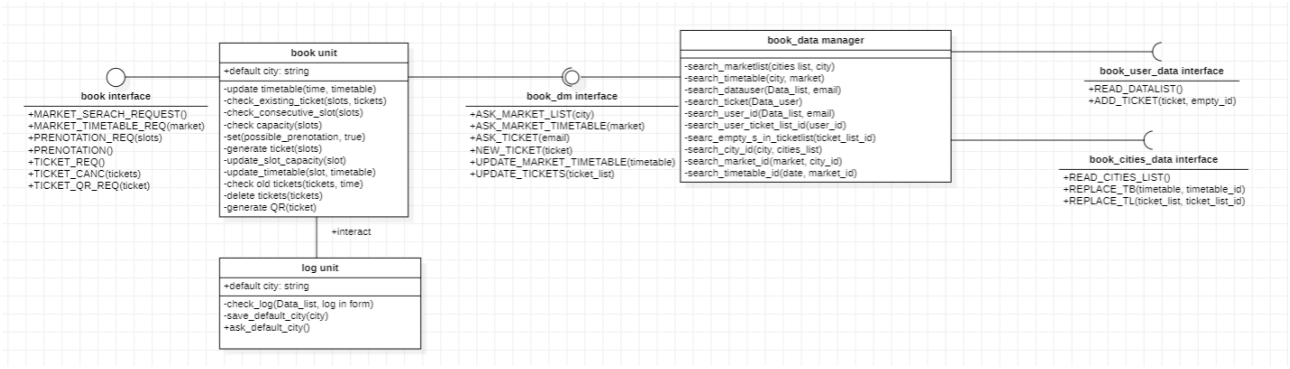


Figure 7- Book system

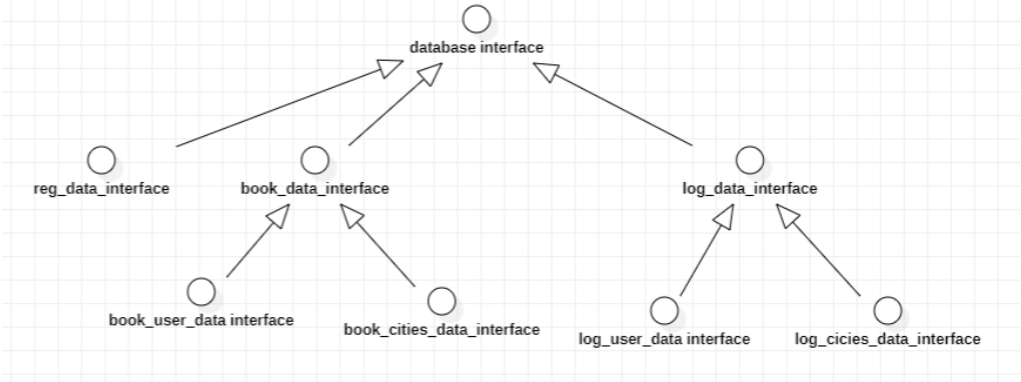


Figure 8- Database interface

2.3.1 Component view – alternative solution

In the previous internal modular structure, some sub-components share the same functions.
A different architectural choice can be used, trying to avoid this redundancy.
One could decide to modularize the Application logic and the Data manager in parts, collecting similar functions.

This way, the implementation will be easier, however we will not have any more detached modules for detached services.

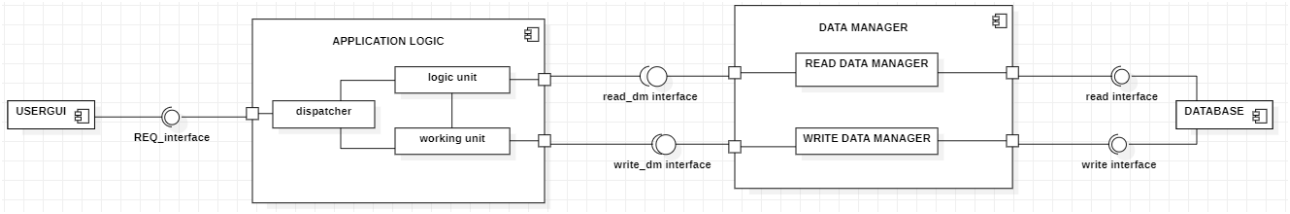


Figure 9

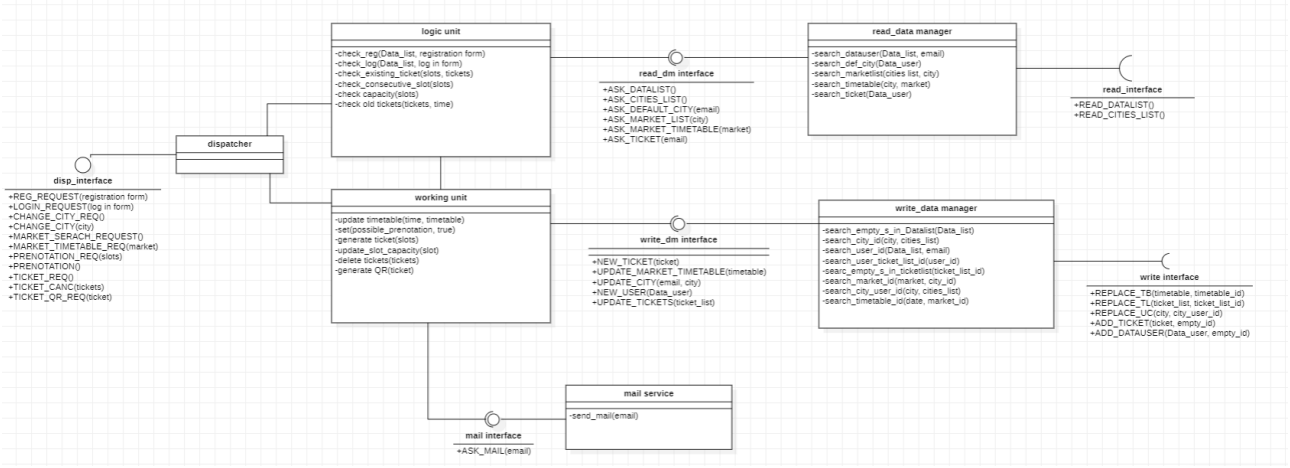


Figure 10

The Application logic is divided as follow:

- Dispatcher:
  brings the User's quest to the right unit

- Logic unit:
  Perform all the comparison operations, like checking the login form or the registration form

- Working unit:
  Perform the remaining operations.

The Data manager is divided as follow:

- Read data manager:
  read and search specified information in the dataset, regardless of its purpose

- Write data manager:
  write and update specified information in the dataset, regardless of its purpose

2.4 Sequence diagrams

In this section, the dynamic behaviour of the System is described by showing the interactions between the various components. The component interfaces will be treated in detail in the next sections.

2.4.1 Registration

In the diagram below is presented the workflow that is executed during the registration of a customer. Once the registration form has been compiled and submitted, the 'Registration unit' checks if there are already similar registrations, by comparing the registration form with all the stored forms in dataset. If the check is ok, the 'Registration unit' update the Data_list by adding the new form and asks the 'Reg_data manager' to store it.

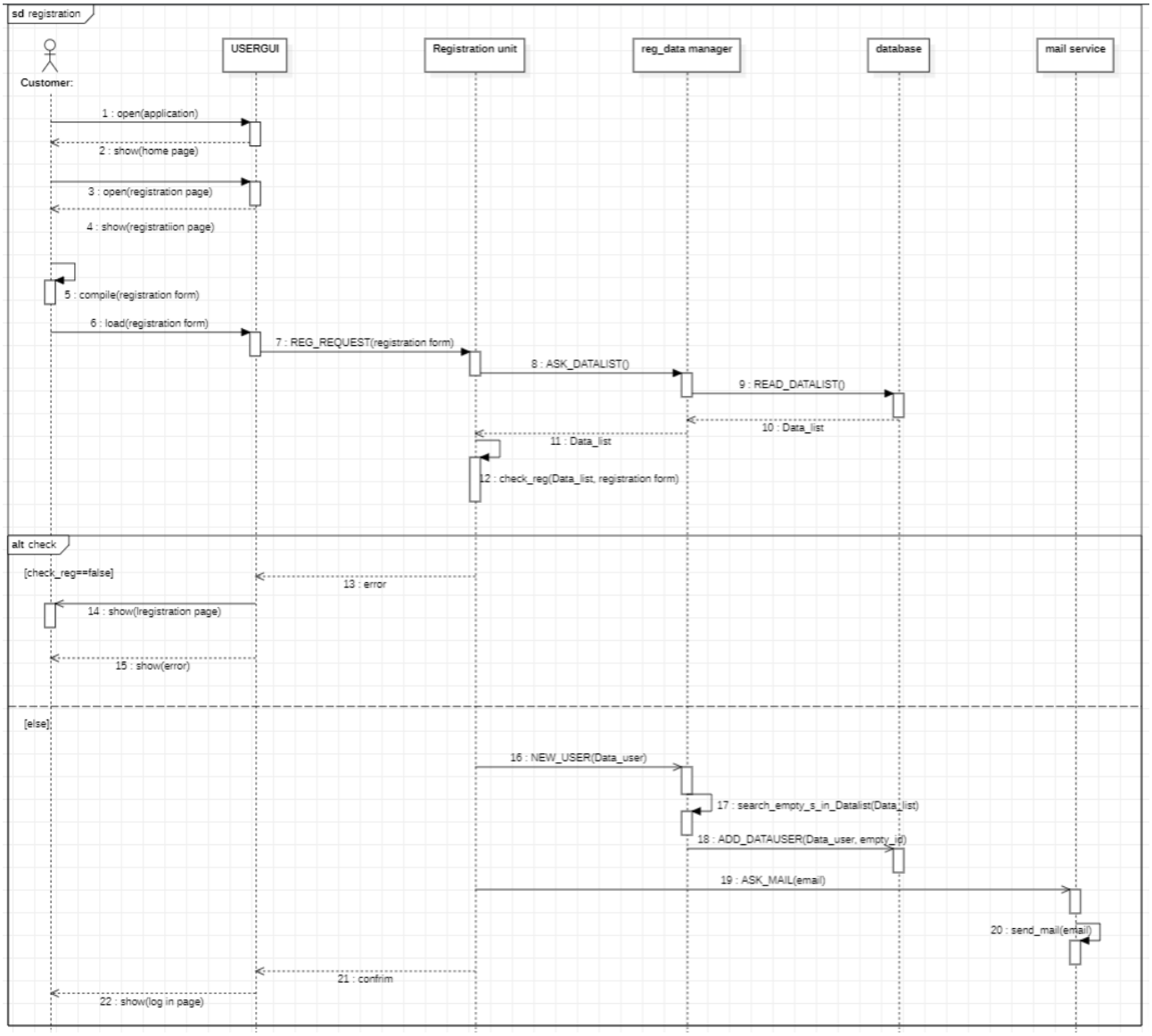


Figure 11

2.4.2 Log in

In the diagram below is presented the workflow that is executed during the log in of a user. Once the login form has been compiled and submitted, the 'Log unit' check if the credentials are correct, by comparing the log in form with all the stored forms in dataset. If the check is ok, the 'Log unit' saves the preferred user's city and directs the user to the booking page.

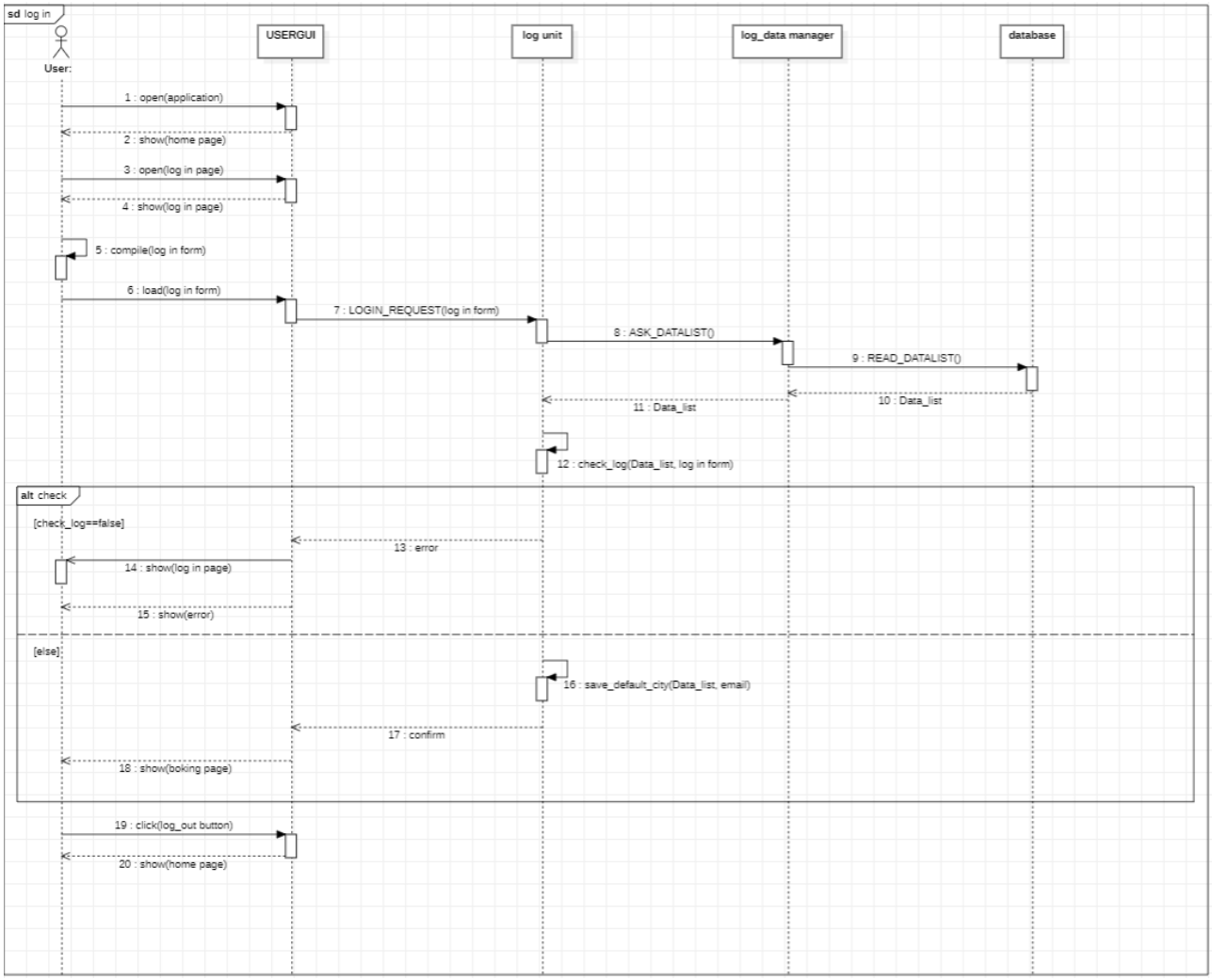


Figure 12

2.4.3 Change default city

The following diagram represents the workflow that occurs when a User wants to change city to find other supermarkets.
By opening the "change city" section, the cities list is shown. Once a new city has been chosen, the Log unit save the default city, ask the corresponding data user to the data manager, and update it. Finally, the modifications are stored and saved.

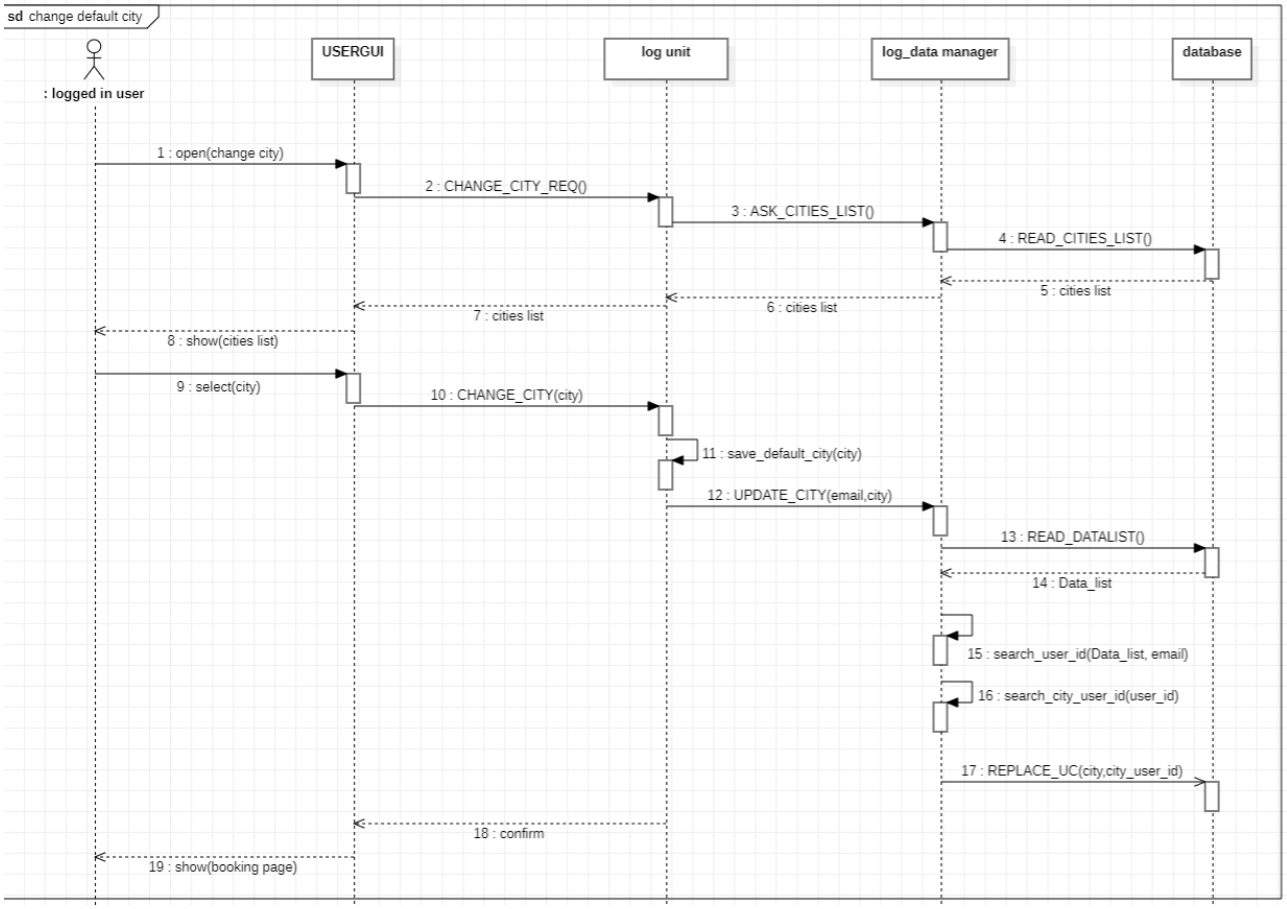


Figure 13

2.4.4 Booking

The following diagram represents the workflow that occurs when a User want to make a reservation.
Firstly, the 'Book unit' asks the 'Log unit' for the user's information; then, thanks to the use of the 'book data manager', a market list is shown.
Once a market has been selected, the book data manager search and send the corresponding market timetable.
This one is firstly updated by the 'Book unit', then, sent to the users.
Once the slots are chosen, if the reservation is feasible and confirmed, the 'Book unit' create the tickets and update the market timetable.
Finally, the modifications and the ticket are stored by the 'Book data manager'.

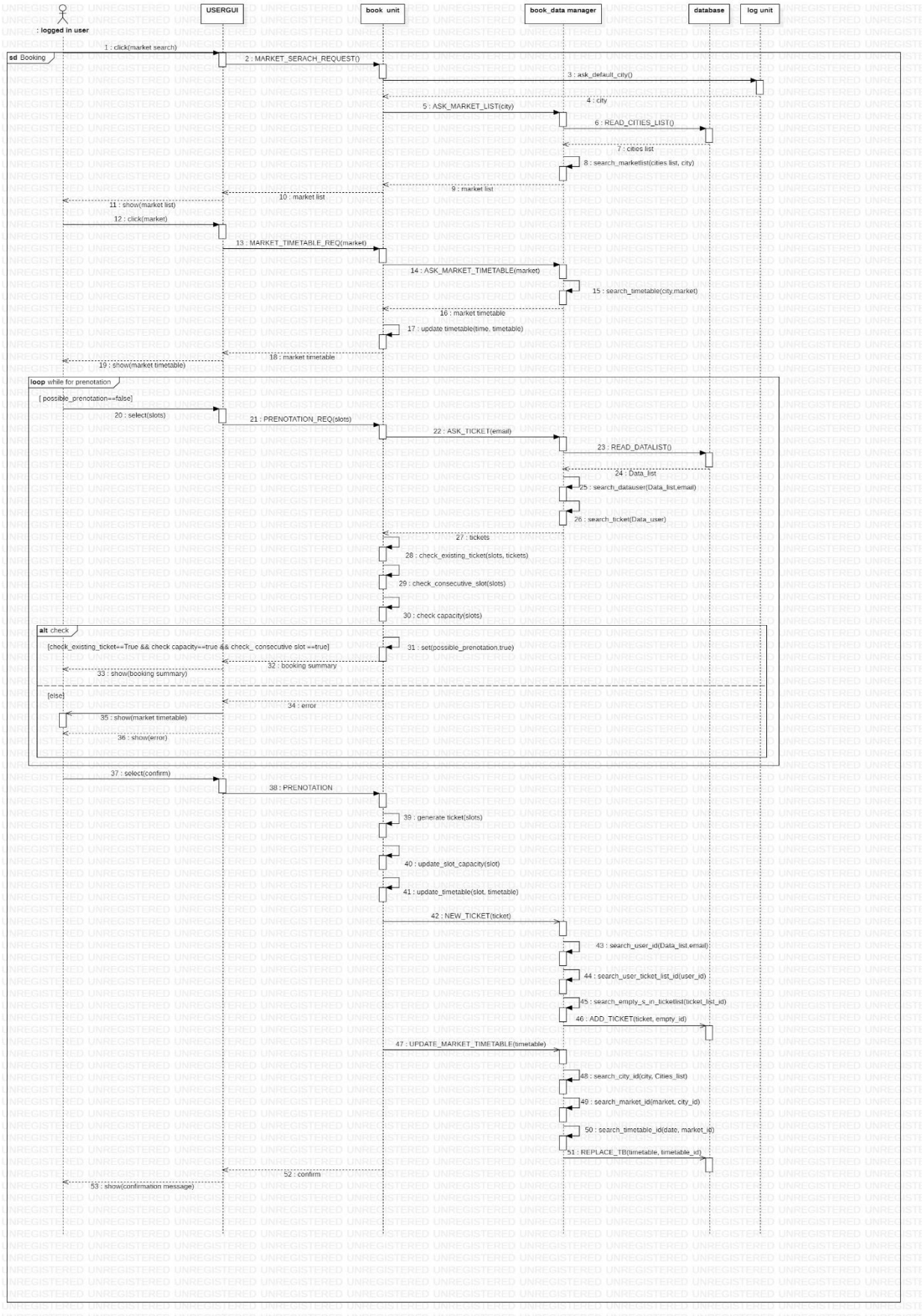

: logged in user
USERGUI
book_unit
book_data manager
database
log unit
sd Booking
1 : click(market search)
2 : MARKET_SERACH_REQUEST()
3 : ask_default_city()
4 : city
5 : ASK_MARKET_LIST(city)
6 : READ_CITIES_LIST()
7 : cities list
8 : search_marketlist(cities list, city)
9 : market list
10 : market list
11 : show(market list)
12 : click(market)
13 : MARKET_TIMETABLE_REQ(market)
14 : ASK_MARKET_TIMETABLE(market)
15 : search_timetable(city,market)
16 : market timetable
17 : update timetable(time, timetable)
18 : market timetable
19 : show(market timetable)
loop while for prenotation
[possible_prenotation==false]
20 : select(slots)
21 : PRENOTATION_REQ(slots)
22 : ASK_TICKET(email)
23 : READ_DATALIST()
24 : Data_list
25 : search_datauser(Data_list,email)
26 : search_ticket(Data_user)
27 : tickets
28 : check_existing_ticket(slots, tickets)
29 : check_consecutive_slot(slots)
30 : check capacity(slots)
alt check
[check_existing_ticket==True && check capacity==true && check_ consecutive slot ==true]
31 : set(possible_prenotation,true)
32 : booking summary
33 : show(booking summary)
[else]
34 : error
35 : show(market timetable)
36 : show(error)
37 : select(confirm)
38 : PRENOTATION
39 : generate ticket(slots)
40 : update_slot_capacity(slot)
41 : update_timetable(slot, timetable)
42 : NEW_TICKET(ticket)
43 : search_user_id(Data_list,email)
44 : search_user_ticket_list_id(user_id)
45 : search_empty_s_in_ticketlist(ticket_list_id)
46 : ADD_TICKET(ticket, empty_id)
47 : UPDATE_MARKET_TIMETABLE(timetable)
48 : search_city_id(city, Cities_list)
49 : search_market_id(market, city_id)
50 : search_timetable_id(date, market_id)
51 : REPLACE_TB(timetable, timetable_id)
52 : confirm
53 : show(confirmation message)

Figure 14

2.4.5 Ticket Validation

In the diagram below is presented the workflow that is executed when a user wants to validate an already existing ticket.
The 'Book user' retrieves all the user tickets (thanks to the book data manager).
Once a ticket has been selected, a Qr code is generated and shown to the market employee.

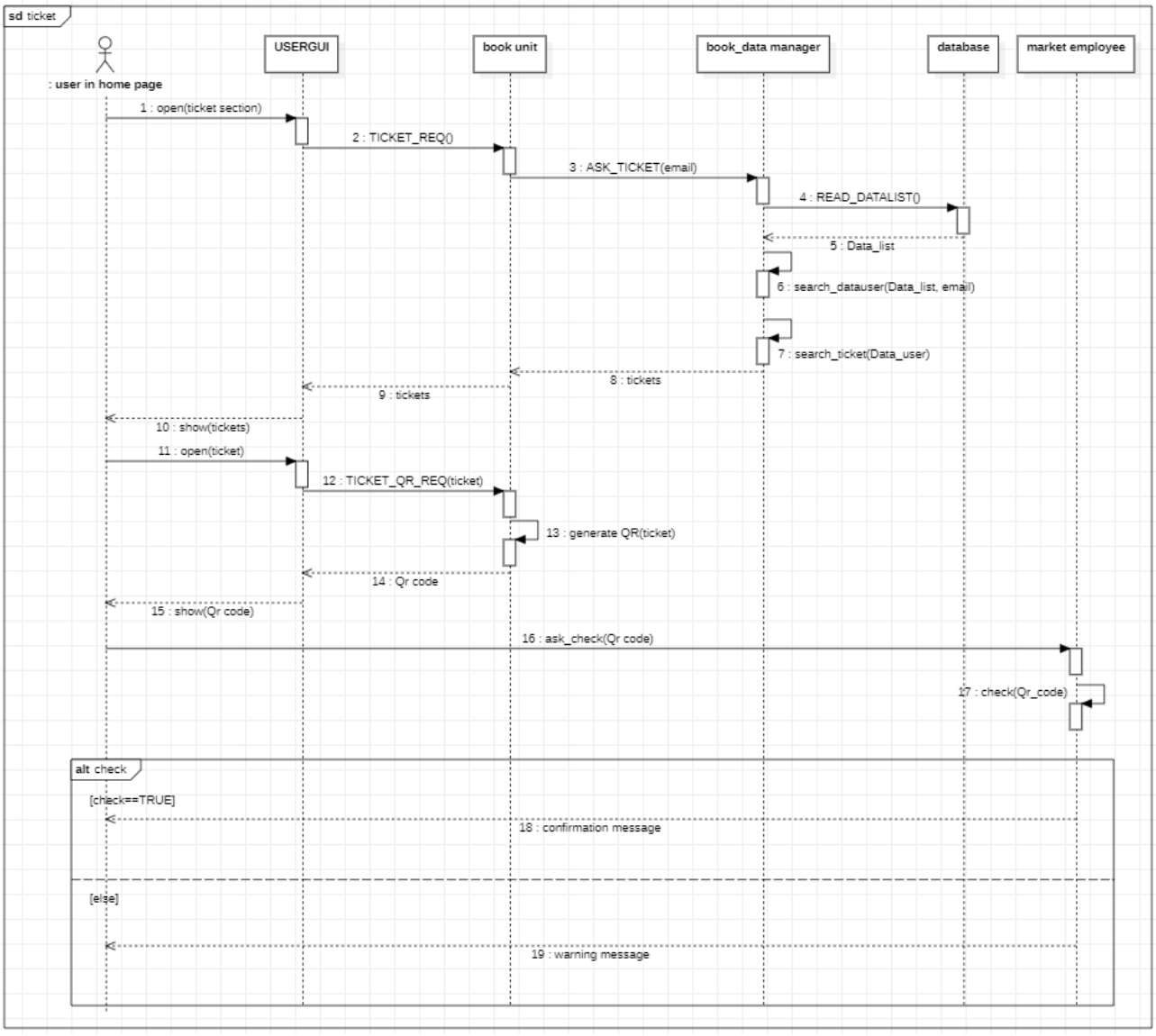


Figure 15

2.4.6 Ticket Cancellation

The following diagram represents the workflow that occurs when a User want to cancel a reservation.
The user can select which ticket must be deleted, then, the data of the user are updated by the 'Book unit' and saved by the 'Book data manager'.

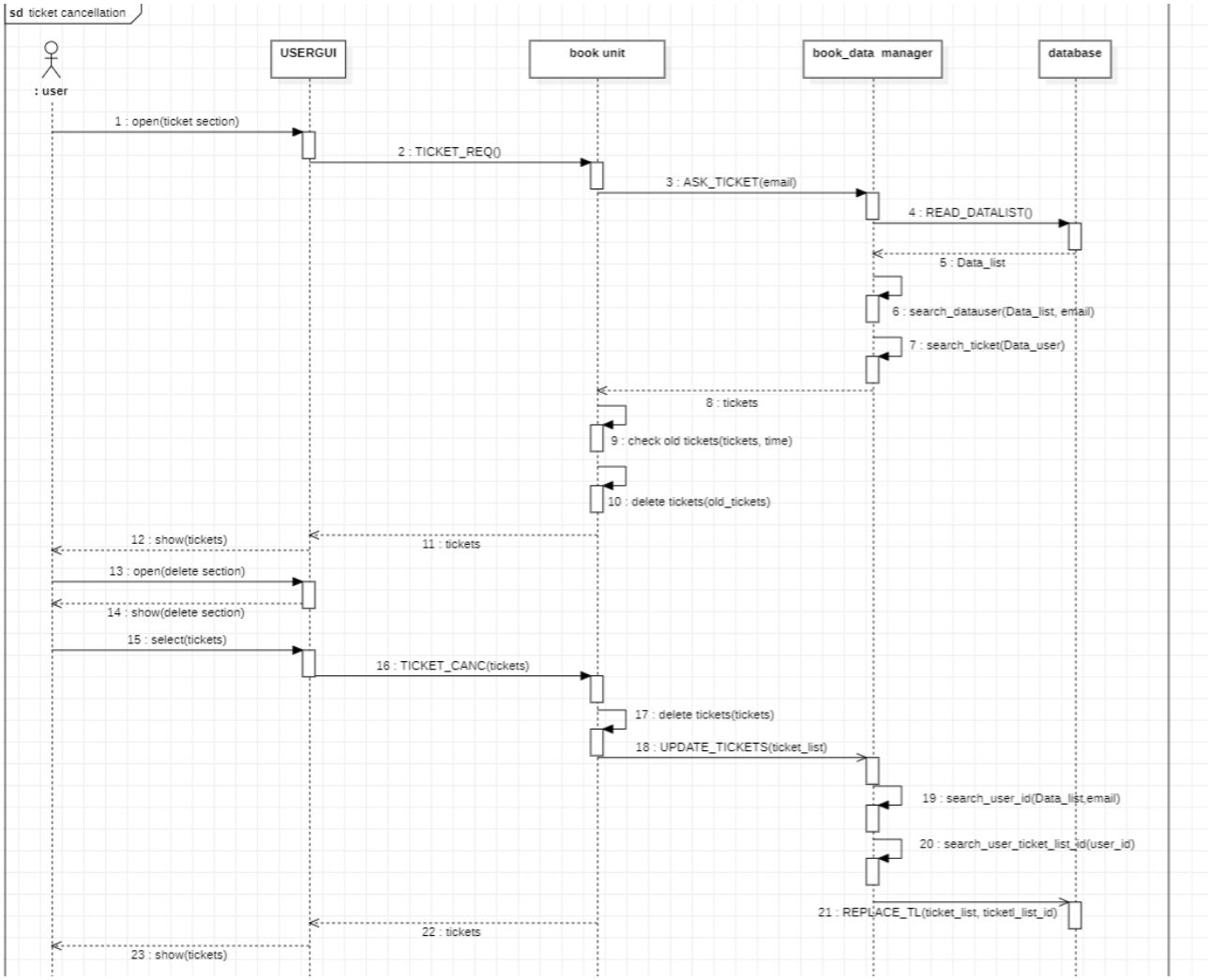
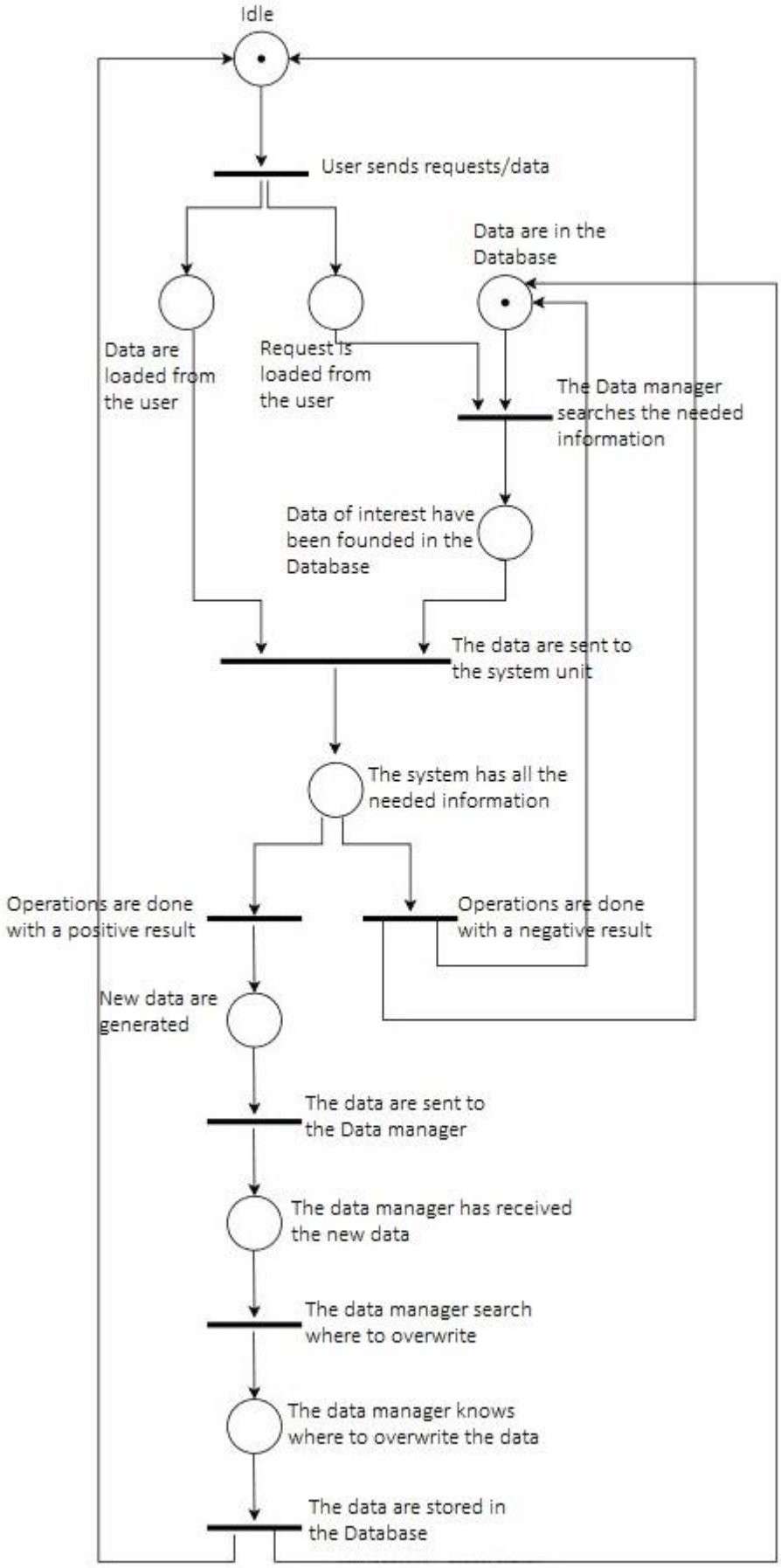


Figure 16

2.5 Petri net

The dynamic evolution of the system, during the main operations like booking or log in, can be represented by a petri net.
It should handle data and requests given by the user and information stored in the Data-base.


Idle
User sends requests/data
Data are in the Database
Data are loaded from the user
Request is loaded from the user
The Data manager searches the needed information
Data of interest have been founded in the Database
The data are sent to the system unit
The system has all the needed information
Operations are done with a positive result
Operations are done with a negative result
New data are generated
The data are sent to the Data manager
The data manager has received the new data
The data manager search where to overwrite
The data manager knows where to overwrite the data
The data are stored in the Database

3 COMPONENT INTERFACES

Modules offer their services through interfaces which act as a contract between a module and its client. This section gives a high-level description of the interfaces that the various system components expose.

3.1 Application logic interfaces

3.1.1 Registration interfaces:

- REG_REQUEST(registration form)
  Generates a new account if the loaded data are correct, otherwise, sends an error message

3.1.2 Log interfaces:

- LOGIN_REQUEST(log in form)
  Handles the authentication process

- CHANGE_CITY_REQ()
  Shows the user a list of city

- CANGE_CITY(city)
  Saves the selected city as default city

3.1.3 Book interfaces:

- MARKET_SEARCH_REQUEST()
  Shows the user a list of markets in the default city

- MARKET_TIMETABLE_REQ(market)
  Shows the user all the available slots in the selected market

- PRENOTATION_REQ(slots)
  Handles the booking process and check if the reservation is feasible

- PRENOTATION()
  Generates the ticket with the previous selected slots

- TICKET_REQ()
  Shows the user his tickets

- TICKET_CANC(tickets)
  Cancels the selected tickets

- TICKET_QR_REQ(ticket)
  Generates a QR code for the selected ticket

3.2 Data manager interfaces

3.2.1 Reg_dm interfaces:

- ASK_DATALIST()
  It returns data related to the Users

- NEW_USER(Data_user)
  It stores data related to the new Users

3.2.2 Log_dm interfaces:

- ASK_DATALIST()
  It returns data related to the Users

- ASK_DEFAULT_CITY(email)
  It returns the default city of the specified User

- ASK_CITIES_LIST()
  It returns data related all the cities

- UPDATE_CITY(city, email)
  It changes the default city of the specified User

3.2.3 Book_dm interfaces:

- ASK_MARKET_LIST(city)
  It returns the markets data of the specified city

- ASK_MARKET_TIMETABLE(market)
  It returns the timetable of the specified market

- ASK_TICKET(email)
  It returns the tickets of the specified User

- UPDATE_TICKETS(ticket_list)
  It stores modification in the ticket list of the User

- NEW_TICKET(ticket)
  It add a new ticket in the user’s ticket list

- UPDATE_MARKET_TIMETABLE(timetable)
  It stores modification of the slots in the specified timetable

3.3 Database interfaces

3.3.1 Reg_data interfaces:

- READ_DATALIST()
  Reads data related to the Users

- ADD_DATAUSER(Data_user, empty_id)
  Writes data related to the new subscribed User

3.3.2 Log_data interfaces

- READ_DATALIST()
  Reads data related to the Users

- READ_CITIES_LIST()
  Reads data related to the cities and markets

- REPLACE_UC(City, city_user_id)
  Saves the new preferred city of the user in the database.

3.3.3 Book_data interfaces:

- READ_DATALIST()
  Reads data related to the Users

- READ_CITIES_LIST()
  Reads data related to the cities and markets

- ADD_TICKET(ticket, empty_id)
  Stores the new generated user ticket in the right section

- REPLACE_TB(timetable, timetable_id)
  Replaces the old timetable with the new updated one

  REPLACE_TL(ticket_list, ticket_list_id)
  Replaces the old user's ticket_list with the new updated one

3.4 Mail interfaces:

- ASK_MAIL(email)
  It handles the confirmation mail services

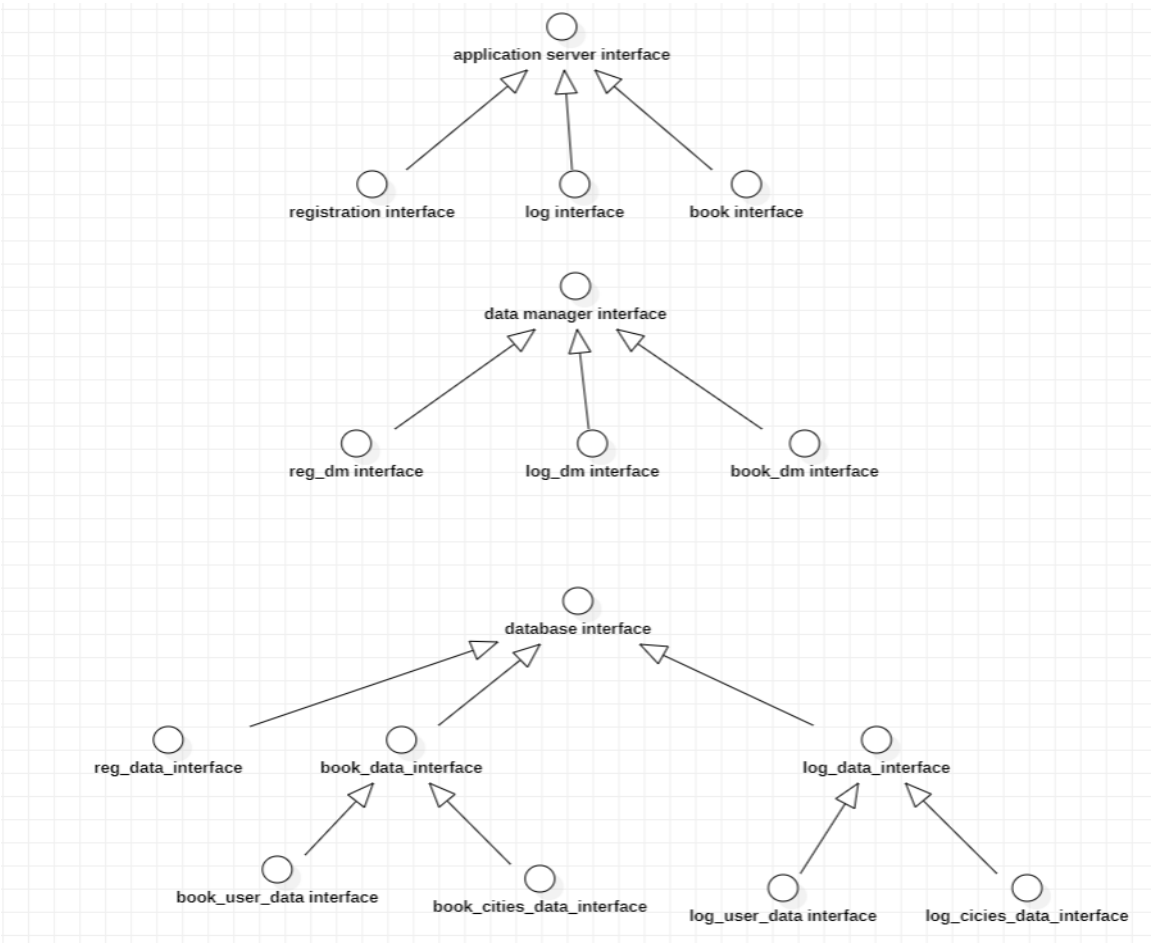


Figure 17

4 COMPONENT OPERATIONS

In the previous section we analysed the visible attributes of the classes in terms of interfaces. Now, we give a brief description of the private operations of the modules.

4.1 Application logic

4.1.1 Registration unit

- Check_reg(Data_list, registration form)
  Compare if there exist an analogous registration form in the data list

4.1.2 Log unit

- Check_log(Data_list, log in form)
  Compare the given credentials with the stored one

- Save_default_city(city)
  Save the given city in an internal attribute

4.1.3 Book unit

- Update_timetable(time, timetable)
  Compare the current time with the time slot time, lock the Past time slots.

- Check_existing_ticket(slots, tickets)
  Compare the selected slot with the User tickets to avoid duplicates in bookings

- Check_consecutive_slot(slots)
  Check if the selected slots are consecutive in time

- Check_capacity(slots)
  Check if the selected slots are not saturated

- Set(possible_prenotation, true)
  Update an internal state

- Generate_ticket(slots)
  Create a ticket with the given slots

- Update_slot_capacity(slot)
  Decrease or increase the slot capacity

- Update_timetable(slots, timetable)
  The timetable is updated with the given slots with modified capacity

- Check_old_tickets(tickets, time)
  Find the expired tickets

- Delete tickets(tickets)
  Cancel the selected tickets

- Generate QR(ticket)
  Create a Quick Response code using the information of the selected ticket

4.2 Data Manager

4.2.1 Reg_data manager

- Search empty_s_in_Datalist(Data_list)
  Find where to store the new user information

4.2.2 Log_data manager

- Search_datauser(Data_list, email)
  Find the userdata in the datalist using the email

- Search_def_city(Data_user)
  Find the preferred city in the userdata

- Search_city_user_id(user_id)
  It return the city-user location in the database

- Search_user_id(Data_list, email)
  It return the user location in the database exploiting the email

4.2.3 book_data manager

- Search_marketlist(cities_list,city)
  Find the city in the cities list

- Search_timetable(city,market)
  Find the timetable of the selected market

- Search_datauser(Data_list,email)
  Find the userdata in the datalist using the email

- Search_ticket(Data_user)
  Find all the stored tickets of the users

- Search_user_id(Data_list, email)
  It return the user location in the database exploiting the email

- Search_city_id(city, cities list)
  It return the location of the city in the database

- Search_user_ticket_list_id(User_id)
  It return the location of the user ticket list in the database

- Search_empty_s_in_ticketlist(ticket_list_id)
  It return the location of the first empty space in the ticket list

- Search_market_id(market, city_id)
  It return the location of the specified market in the database

- Search_timetable_id(date, market_id)
  It return the location of the specified timetable in the database exploiting the current date

5 GRAPHICAL USER INTERFACE

The User Interfaces are reported in the “Specific Requirements” chapter in the RASD. In this section a detailed statechart related to the graphical interface is provided.

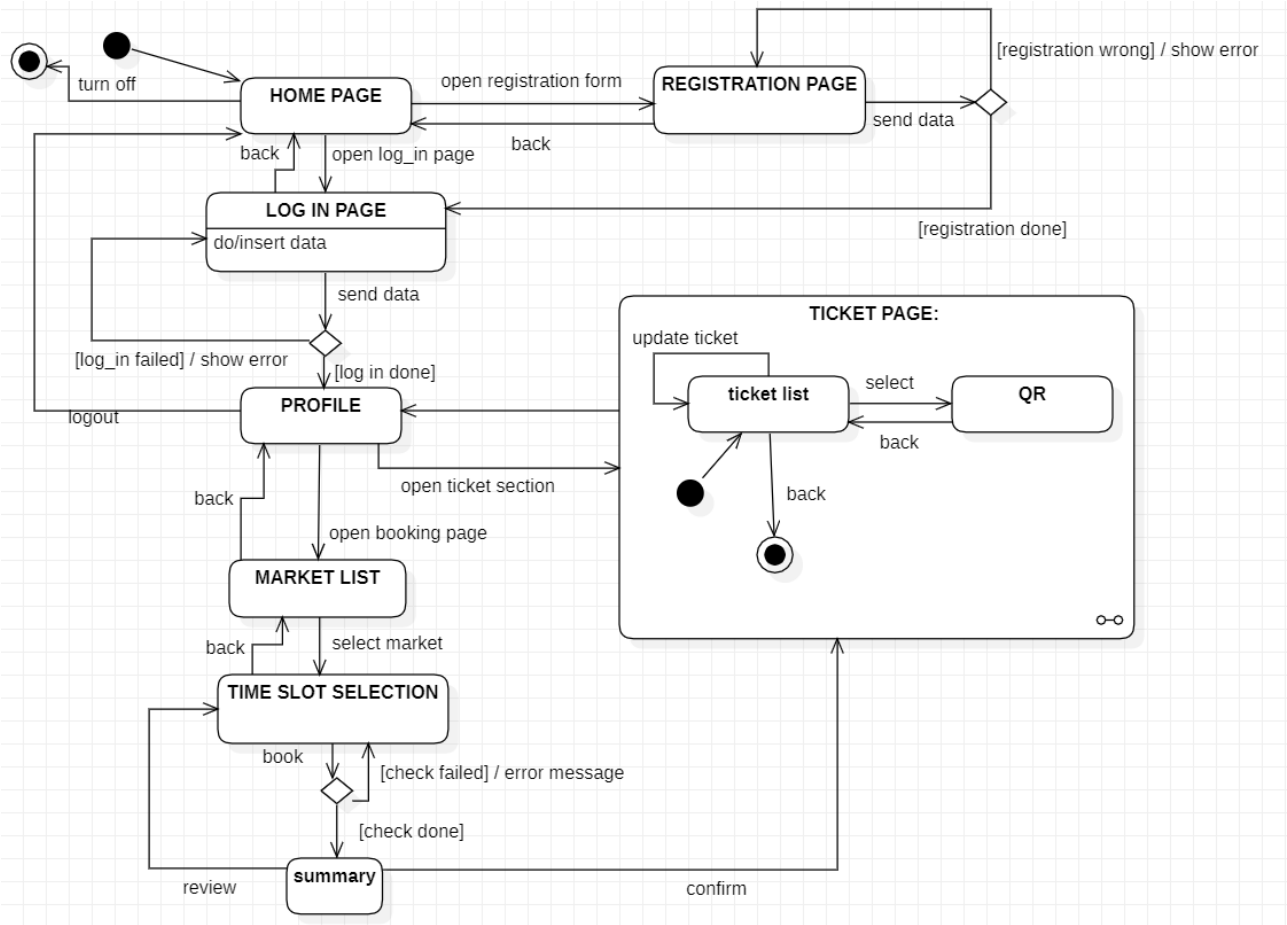


Figure 18

6 REQUIREMENT TRACEABILITY

In this chapter , the mapping between the requirement presented in the RASD and the modules presented in the DD is reported.

6.1 Functional Requirements – Core

ID	Requirement	Modules
R1	The system must allow users to sign up if and only if they are new.	Registration unit Reg_data manager Database
R2	The system must allow users to sign in if and only if already registered and the credentials inserted are valid.	Log unit Log_data manager Database
R3	The system must allow the user to log out.	Log unit
R4	The system must allow the user to search and select the supermarket.	Book unit Book_data manager Database
R5	The system must allow the user to choose the city.	Log unit Log_data manager Database
R6	The system must restrict the supermarket research to the ones that are in the user city	Book unit Book_data manager Log unit Database
R7	The system shows time slots of the selected market according to available timetable and crowding data.	Book unit Book_data manager Database
R8	The system must allow the user to select the day on which they want to book in the selected supermarket, among the ones that the store manager makes available.	Book unit
R9	The system allows the user to make a reservation only in free time slots	Book unit
R10	The system must generate one and only one ticket per reservation.	Book unit
R11	The system must update the time slot crowding of the supermarket for each ticket created/deleted in that store.	Book unit Book_data manager Database
R12	Each ticket generated by the system must have a unique QR code	Book unit
R13	The system allows the user to make a reservation over more time slots only if they are all free and in the same day.	Book unit
R14	The system allows the user to make a reservation only if it does not contain time slots already involved in existing reservations at the selected market on the same day	Book unit

R15	the system must not allow the user to have concurrent reservations over a maximum number.	Book unit Book_data manager Database
R16	The system must allow the user to check their pending tickets	Book unit Book_data manager Database
R17	The system must allow the user to delete their pending tickets	Book unit Book_data manager Database
R18	The system does not allow the user to change single settings of already generated tickets.	Book unit
R19	The system must automatically delete the ticket at the end of its reservation time	Book unit Book_data manager Database

6.2 Functional Requirements – Medium priority

ID	Requirement	Modules
R20	The system should allow the user to update their data and delete their account.	Registration unit reg_data manager Database
R21	The system should be able to auto-log in the user if they have previously ticked the "Remember me" box.	Log unit Log_data manager Database
R22	The system should allow the user to recover their credentials.	Registration unit Reg_data manager Database Mail service
R23	The system should alert the user when their account is created/deleted.	Mail service
R24	The system should filter supermarkets by name.	Book unit
R25	The system should remind the user about incoming reservations before their beginning	Log unit Log_data manager Database Mail service

6.3 Functional Requirements – Good to have

ID	Requirement	Modules
R26	The system should allow the user to select the app language.	Usergui
R27	The system should allow the user to contact the assistance.	Usergui Mail service

R28	The system should allow the user to download the QR ticket to use it offline.	Book Unit
R29	The system should show the recent supermarket researches.	Log unit Log_data manager Database
R30	The system should memorize the favorite supermarkets.	Log unit