

M.Sc. Automation and Control Engineering

Software Engineering (for Automation)

Academic year 2021/2022

**Implementation Document**

CLup: Customers Line-up



*Advisor*: Prof. Matteo Giovanni Rossi

*Students*: Alessandro Nocentini [alessandro.nocentini@mail.polimi.it](mailto:alessandro.nocentini@mail.polimi.it)

Alberto Valentini [alberto.valentini@mail.polimi.it](mailto:alberto.valentini@mail.polimi.it)

Fausto Luca Pichierri [faustoluca.pichierri@mail.polimi.it](mailto:faustoluca.pichierri@mail.polimi.it)

**Table of Contents**

1. [Introduction](#_Introduction) 3

1.1 [Purpose](#_1.1__Purpose) 3

1.2 [Scope](#_1.2__Scope) 3

1.3 [Definitions, Acronyms, and Abbreviations](#_1.3__Definitions,) 3

1.3.1 [Definitions](#_1.3.1_Definitions) 3

1.3.2 [Acronyms](#_1.3.2_Acronyms) 3

1.3.2 [Abbreviations](#_1.3.3_Abbreviations) 3

1.4  [References](#_1.4__References) 3

2. Source code structure 4

2.1 Client side 3

2.2 Server side 3

2. Requirements implementation 4

# Introduction

## 1.1 Purpose

The purpose of this document is to present a possible way to realize a prototype of the mobile application “Customer Line-Up“ according to the indications described in the Requirement Analysis and Specification Document and in the Design Document. The aim of the prototype is to be a demo of the app’s features and of how they work.

## 1.2 Scope

The next sections of the documents explain how the project requirements could be implemented, motivating the architecture and the programming language selected and by analyzing the advantages and the drawbacks for each approach. The testing of the application is outlined in the Testing Document.

## 1.3 Glossary

## 1.3.1 Definitions

* Customer: person that has 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.
* Store manager: person that 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 and the user.

* Valid ticket: ticket enabled to access into the supermarket.
* 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.
* DD: Design Document.
* TD: Testing Document.
* CLup: Customer Line-Up (name of the application).

## 1.3.2 Abbreviations

* Rn: Requirement number n.

## 1.4 References

* Requirement Analysis and Specification Document – Customer Line-Up.
* Design Document – Customer Line-Up.
* Testing Document – Customer Line-Up.
* Project proposal: [link to the document](https://docs.google.com/document/d/1ly9NTnhpajnhPrgtFI8vu0EJ9nhfC0wtdk1QYEIhB5U/edit)
* MIT App Inventor: [App Inventor main page](https://appinventor.mit.edu)
* Google Sheets: [Wikipedia source](https://en.wikipedia.org/wiki/Google_Sheets)
* Google Apps Script: [Wikipedia source](https://en.wikipedia.org/wiki/Google_Apps_Script)

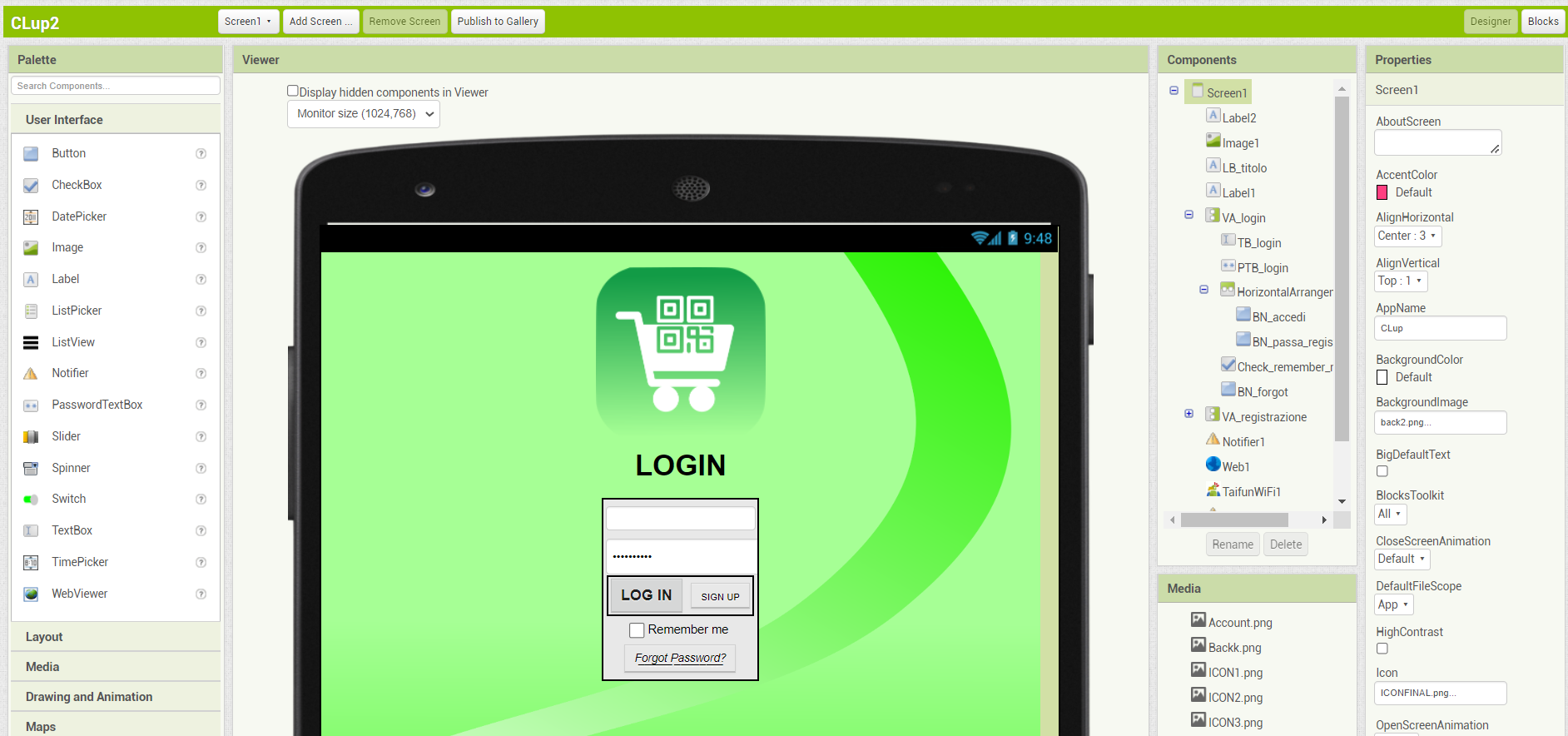
# Code source structure

## 2.1 Client side

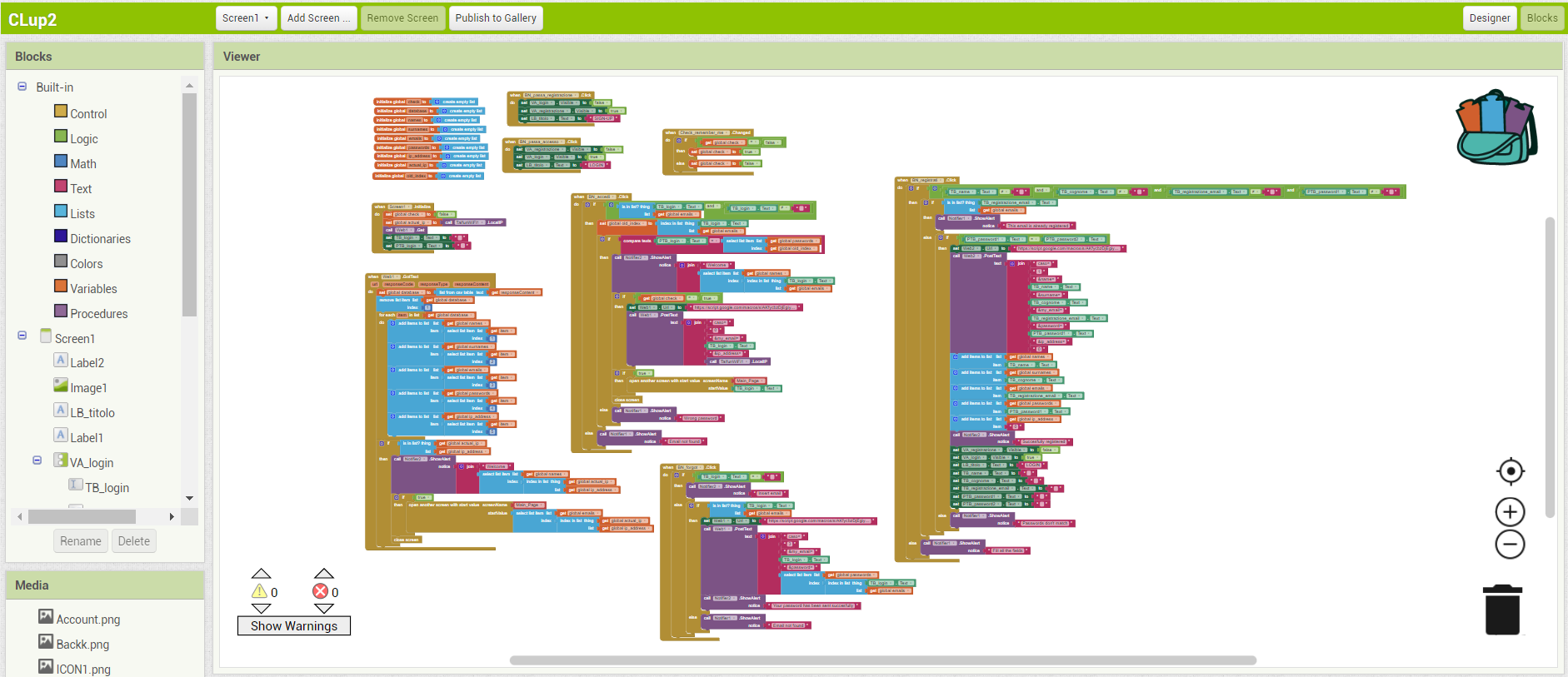
The prototype has been carried out by taking advantage of the web application MIT App Inventor.

It is a developing platform for Android applications by the Massachusetts Institute of Technology, which provides a block-based programming language in such a way to build apps in an easy and quick way owing to a drag and drop approach. On the other hand, the code structure is not robust and with low portability, but more than suitable for prototype use.

The code is organized on "screens". Following a modular approach, each section of the application has its stand-alone piece of code and they communicate one another by exchanging from time to time the information needed to provide the service.



**App Inventor Screen "Designer" view**

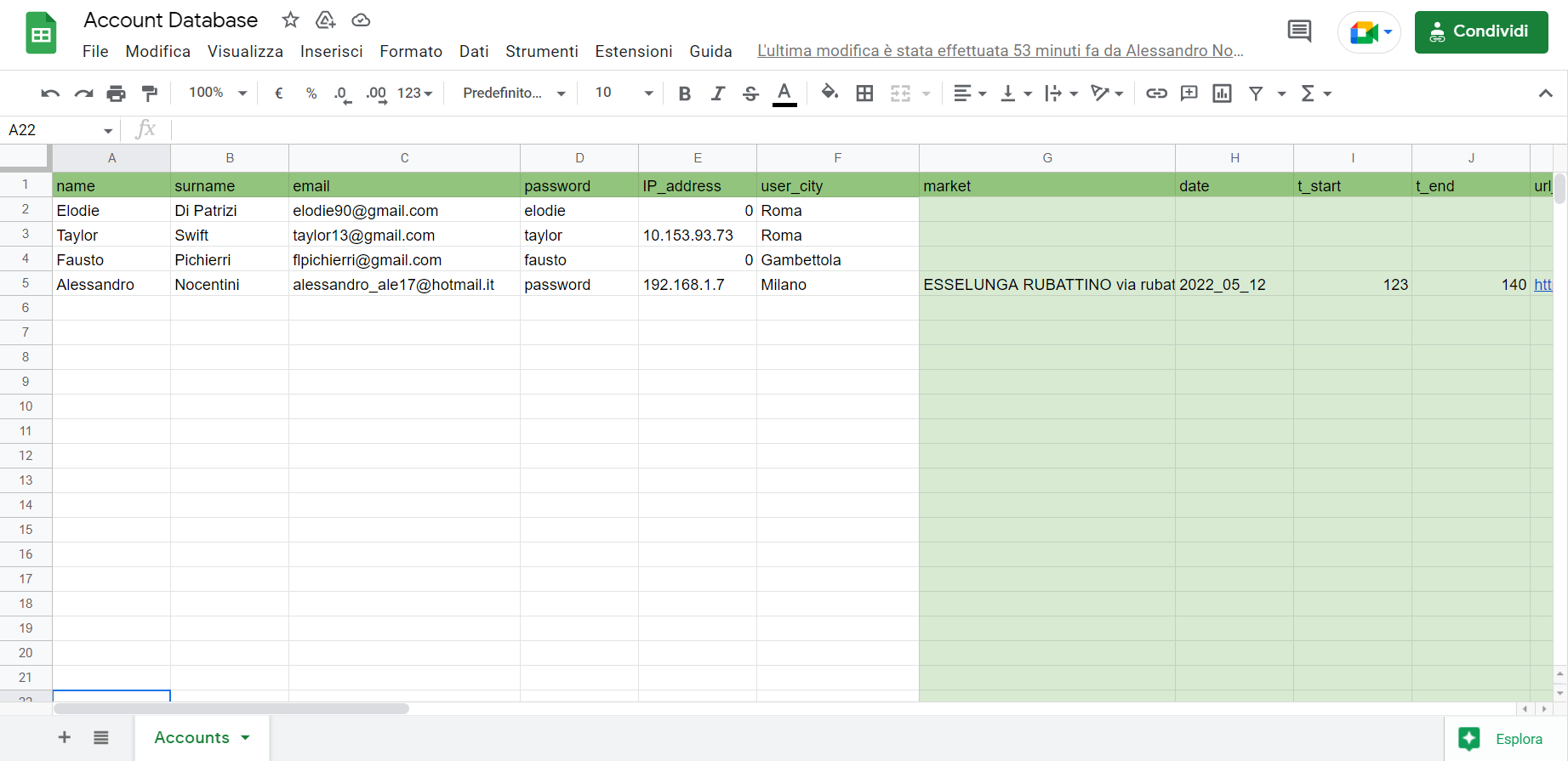


**App Inventor Screen "Blocks" view**

## 2.2 Server side

The application works in combination with an online database through which data are read and stored. Several Google Sheets have been used as database according to the type of information needed to be stored, in order to simplify the flow data managing and the maintainability. The main drawback of this solution is that data are stored on another company server, but for a demo this is acceptable and Google Sheets represent a good solution since they are free and easy to handle with the support of a Google Apps Script.

One main Google Script is used to receive writing requests from the application and, eventually, apply them to the proper Google Sheet. Another Google Script is employed in order to check the current hour and update the data at the stroke of midnight.

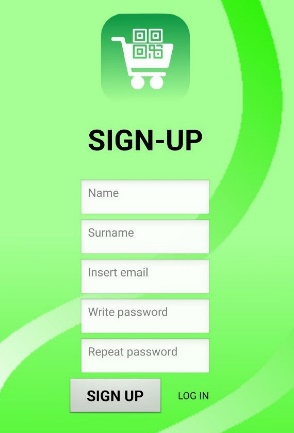


**Google Sheet as database**

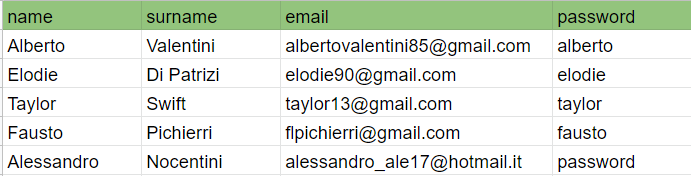
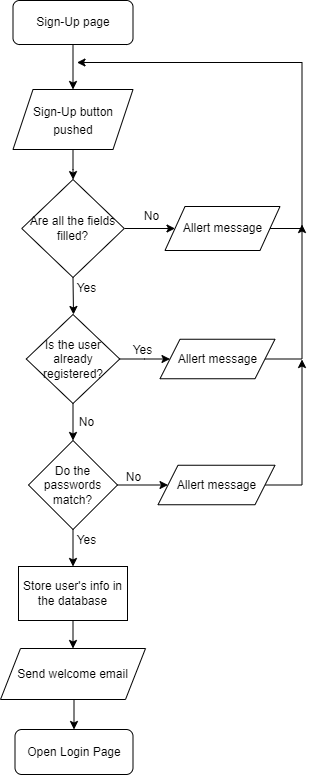
# Requirements implementation

According to section 3.2 of the Requirements Analysis and Specification Document, below are enlisted the requirements of the application, if they have been implemented or not, and, in the positive case, a brief description of the working principle.

* **R1**: The system must allow users to sign up if and only if they are new → *Implemented*

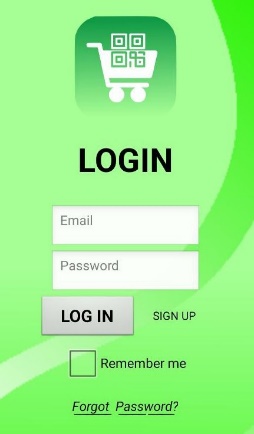
 *\*See R22 for further details about the “welcome e-mail”*

**1.1 Sign-Up user view**

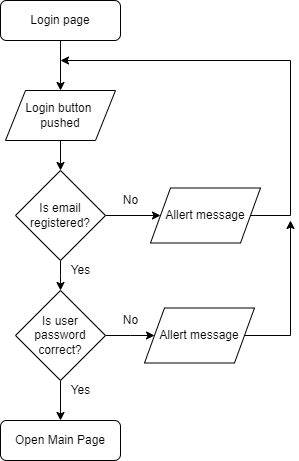


**1.2 Working Principle**

**1.3 Accounts Database**

* **R2**: The system must allow users to sign in if and only if already registered and the credentials inserted are valid → *Implemented*

**2.1 Login user view**

**

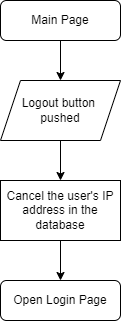
**2.2 Working Principle**

* **R3**: The system must allow the user to log out → *Implemented*

*\*see R20 for further details about the IP address management*

**

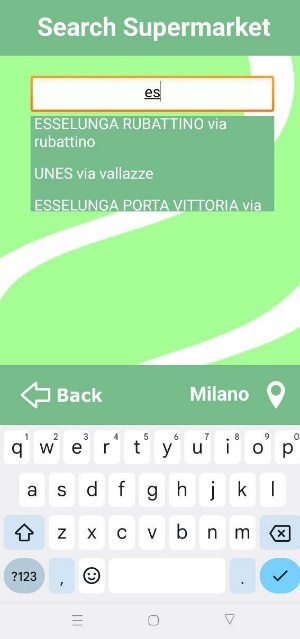
**3.1 Main Page user view**

**

**3.2 Working Principle**

* **R4**: The system must allow the user to search and select the supermarket → *Implemented*

Clicking on the search button from the "main page", the user is redirected to the "search supermarket page "where has been implemented a search bar through which the user can write the market name and then select it from a drop-down menu.

**4. Search Supermarket Page**

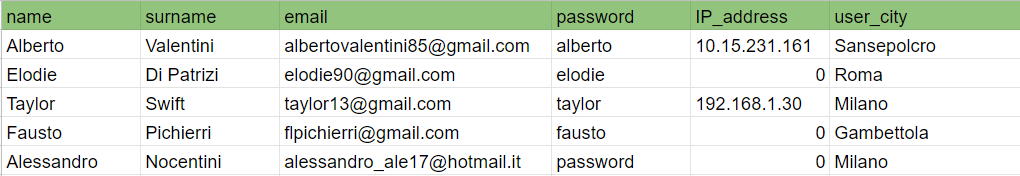
* **R5**: The system must allow the user to choose the city → *Implemented*

The default user's city is "Milano". Clicking on the bottom right-hand side button, the user can choose from a list the city in which they want to search the supermarket. For demo purposes, just four cities are available and only "Roma" and "Milano" are actually implemented.

Once a city is selected, the choice is registered in the database and applied on the "Search Supermarket" page.



**5.1 City Selection**

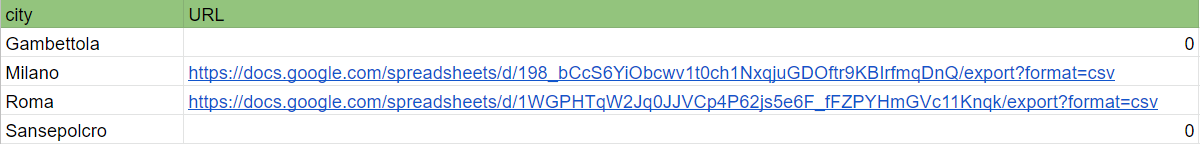


**5.2 Accounts Database**

* **R6**: The system must restrict the supermarket research to the ones that are in the user city

→ *Implemented*

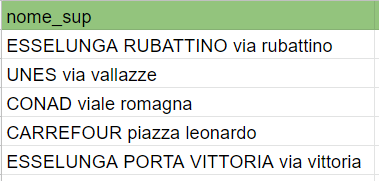
1. Read the user city from the "Account Database".
2. Read the corresponding city URL from the "Cities Database".



**6.1 Cities Database**

3) Use the URL to access the "City Database" (e.g. the below “Milano” Database) where are

stored the city's markets and the research is restricted to only these ones.

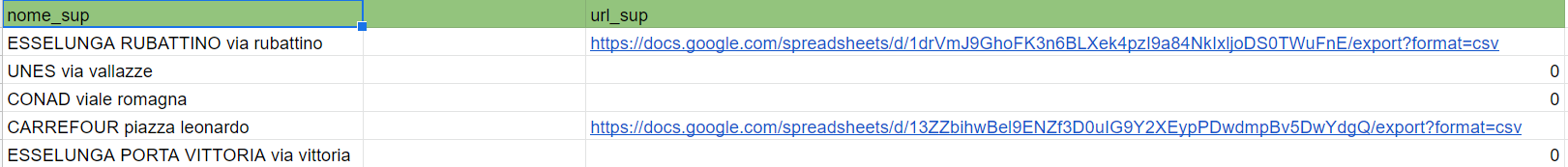


**6.2 “Milano” Database**

* **R7**: The system shows time slots of the selected market according to available timetable and

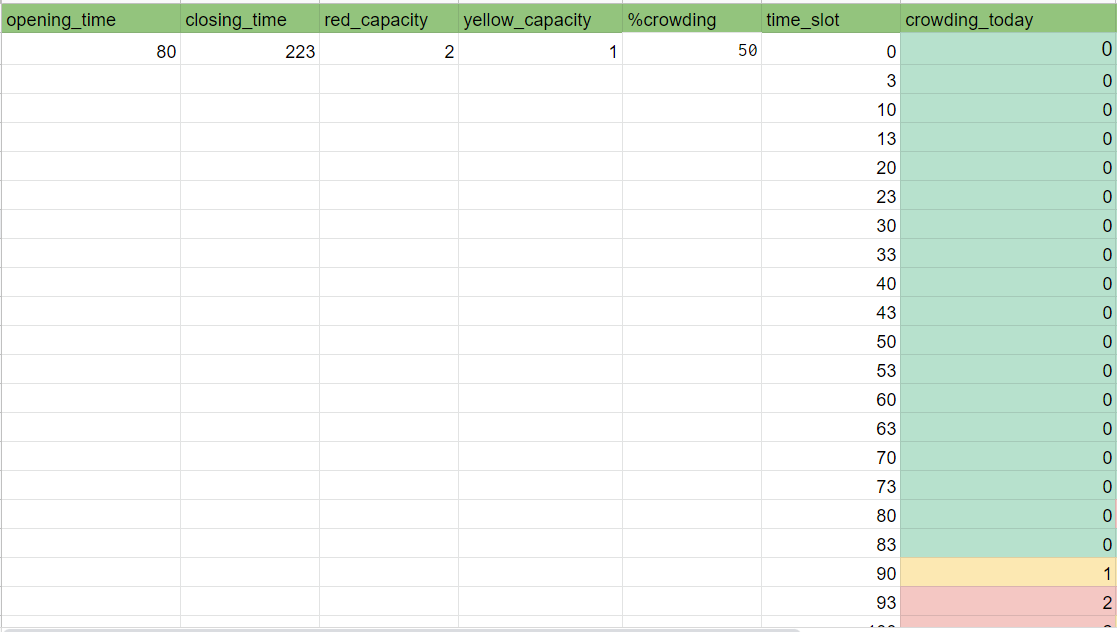
crowding data → *Implemented*

1. Once the user has selected a store, its corresponding URL is read from the "City Database".



**7.1 “Milano” Database**

1. The URL is used to access the "Market Database" (e.g. the below “Esselunga Rubattino” Database). Note: there is a Market Database for each supermarket.



**7.2 “Esselunga Rubattino” Database**

1. In the user interface, the time slots are represented as buttons with impressed the slot start time, so there may be up to a maximum of 48 possible buttons, each one for half an hour.

Then, from the "Market Database" is read the market opening and closing time (in the above example, 80 stands for 8 am, 223 means 10.30 pm, and so on) and all the “button time slots” outside of this range are omitted. Afterward, the time slots inside the market timetable, but expired at the user's current time, are marked in gray.

The crowding levels of the day are registered for each time slot in the "crowding today"

column. If the booking number of a time slot is equal to the maximum capacity set by the

store manager ("red capacity"), then the corresponding button in the UI is displayed as

red to mean "full time-slot". If it is not full but greater than or equal to the warning level

"yellow capacity", it is marked with a yellow background. The other remaining time slots

are represented with white buttons.

The meaning of the buttons and the colors is also explained in a legend, which pop-ups

pushing on the question mark button.

**7.3 "Esselunga Rubattino" timetable 7.4 Legend**

* **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 → *Implemented*

For demo purposes just two days have been made selectable: the current day and the day after, the extension to more days is easily implementable following the same logic.

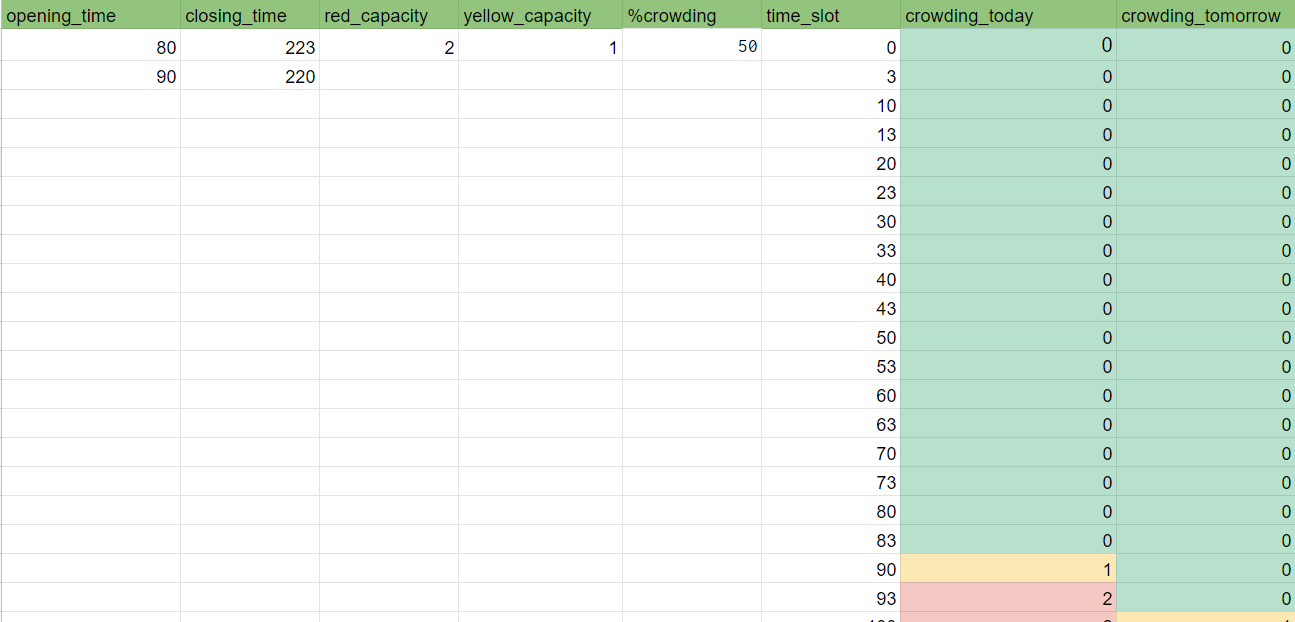
The day switch in the application is possible through the “arrow button” next to the date of the day.

In the database, instead, an additional opening time and crowding column are inserted. Then, every time the booking day is changed, the system does the same operation as described in [R7.3](#R73), but with the corresponding day dataset.

The days crowding data in the database are automatically updated every midnight through a trigger inside a Google App Script: data in tomorrow column are shifted in today column and, then, tomorrow column is reset.



**8.1 "Esselunga Rubattino" day after timetable**



**8.2 "Esselunga Rubattino" Database complete view**

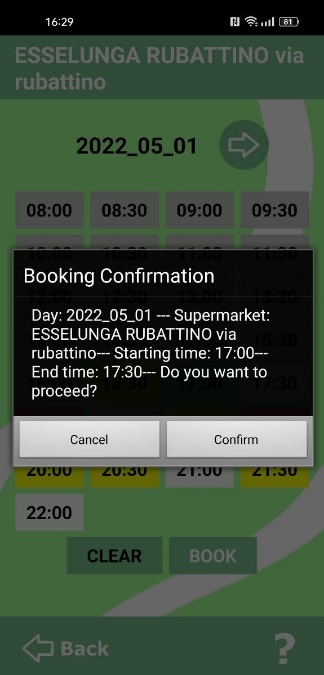
* **R9**: The system allows the user to make a reservation only in free time slots → *Implemented*

If the user tries to click on one of the not selectable grey and red buttons, nothing happens. Instead, if a free time slot is selected (a yellow or white button), its background changes into light blue and the "book" and "clear" buttons appear.

  **9.1 Anything selected 9.2 Free time slot button selected**

* **R10:** The system must generate one and only one ticket per reservation → *Implemented*

"Clear" just deselects the buttons. Clicking on "Book", instead, the system pops up a message asking to the user if they want to proceed with the reservation.

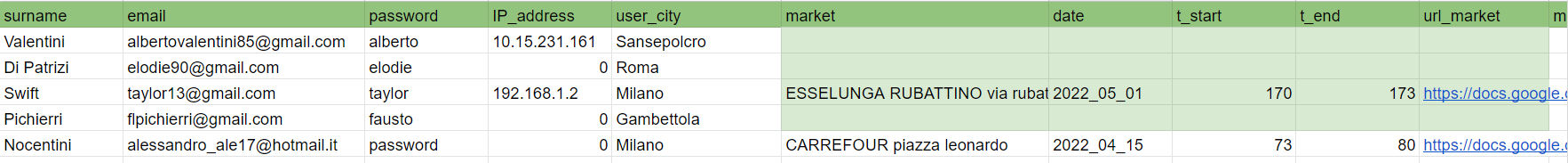
 

**9.3 Booking Confirmation 9.4 Ticket in user view**

"Cancel" brings the user back. if there are no incompatibilities with already existing user tickets in the database (see R14 and R15 for exceptions), clicking on “Confirm”, instead, the ticket is created.

In the Account Database, the following reservation info are stored: market's name, reservation day, start and end time of the booked time slot, market’s database URL. This URL will be then used to update crowding data when the ticket is deleted.

User tickets are stored in the first available position on the user's account row.



**9.4 Ticket in Account Database**

* **R11**: The system must update the time slot crowding of the supermarket for each ticket

created/deleted in that store → *Implemented*

Each time a ticket is created, the system increases by one unit the crowding data of the booked time slot in the corresponding Market Database. If the reservation involves more consecutive time slots, the increment is made for each of the time slots in the booking range.

On the other hand, when a user's ticket is deleted, the system uses the URL saved with the ticket in the Account Database to access the corresponding Market Database and, then, the crowding is decreased by one unit in each of the interested time slots.

* **R12**: Each ticket generated by the system must have a unique QR code → *Implemented*

At the moment the user confirms the booking, the system opens a page with inside the QR code that must be scanned at the entrance of the supermarket. It contains some information that allow to uniquely identify the reservation.

**12.1 Ticket page 12.2 QR code scan**

* **R13**: The system allows the user to make a reservation in multiple time slots only if they are

all free,consecutive and in the same day → *Implemented*

In order to make a reservation over more time slots, is not possible to select only the start and end time slots, but the user has to consecutively select all the time slots in-between.

As described before, in fact, the system allows selecting only time slots that are not full or expired, so, in this way, it checks that all the time slots involved in the booking are free.

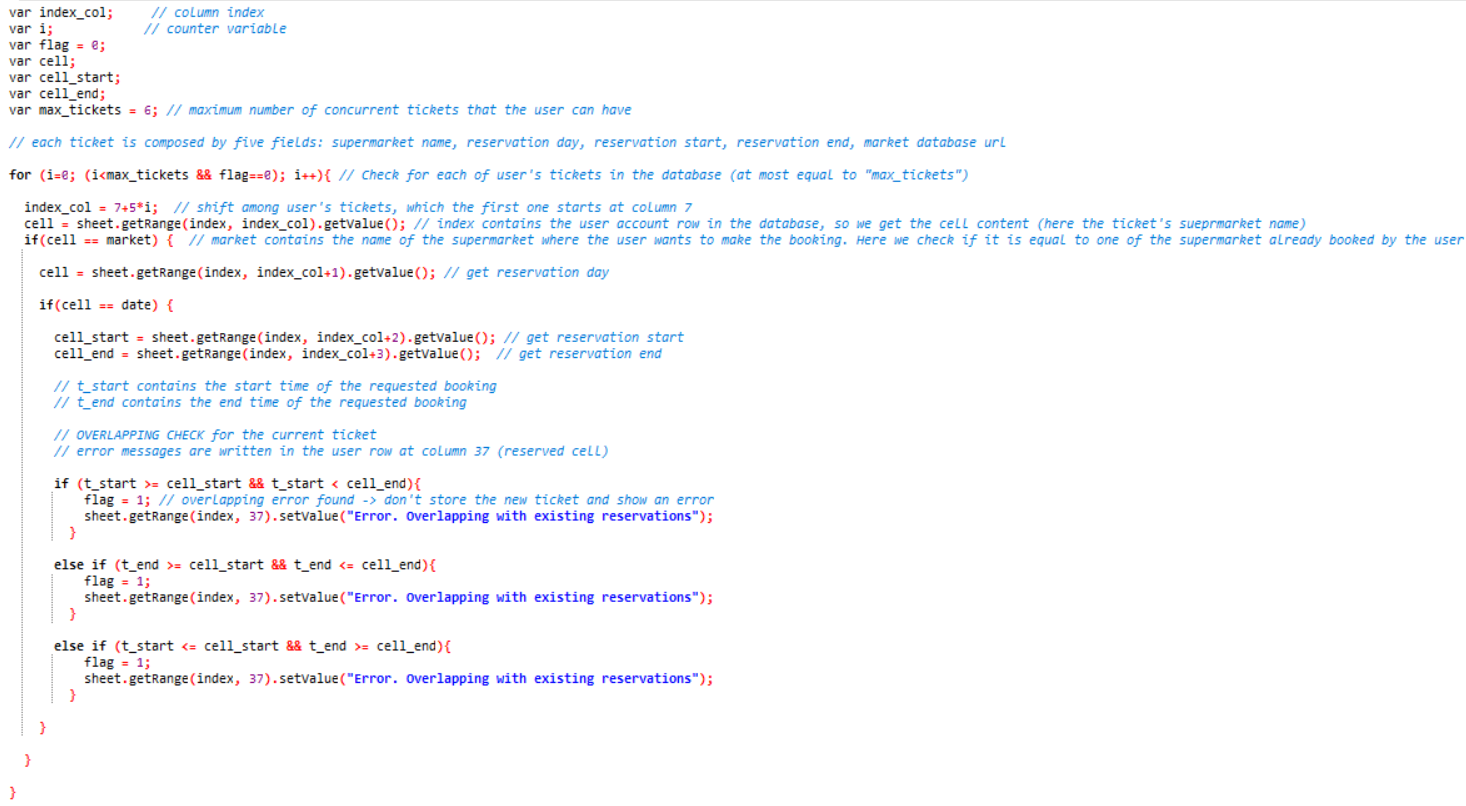
A reservation can't also involve time slots of different days, as when the user changes the booking day, the system resets the selection.

* **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

→ *Implemented*

Every time a booking request is sent to the system, a Google App Script runs the below routine searching for possible overlapping issues with the already existing tickets in the Account Database and, in that case, writes an error message in a reserved cell, to be then displayed to the user by the application.



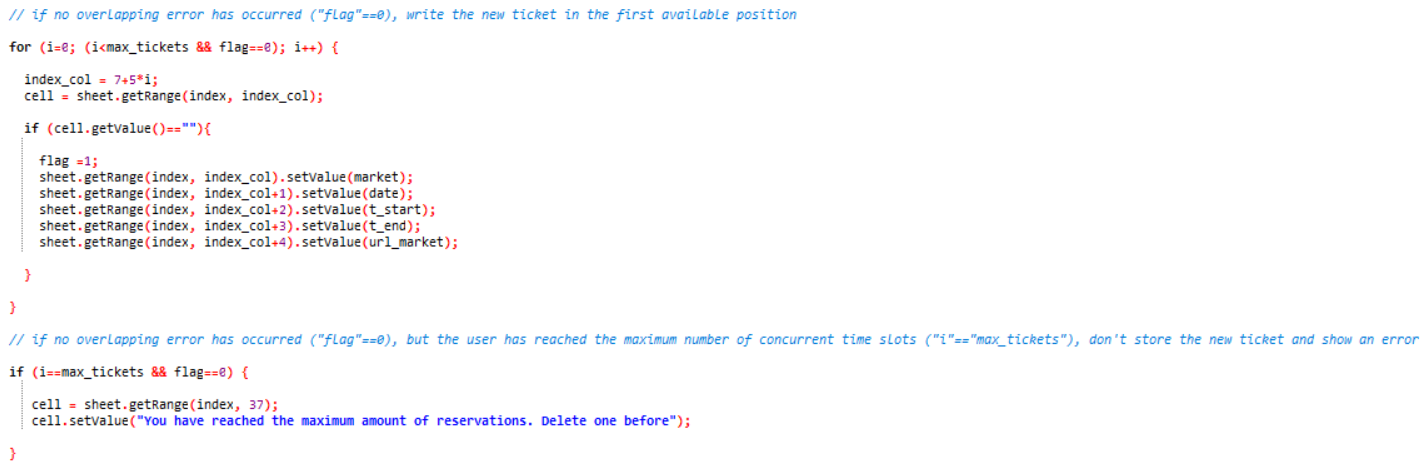
**14.1 Overlapping error routine**

* **R15**: the system must not allow the user to have concurrent reservations over a maximum number

→ *Implemented*

If the system has not previously detected overlap errors, then it checks if the user has a ticket slot available. In the positive case, the system stores the new ticket in the Account Database, otherwise it shows an error message.

In the prototype, the maximum number of concurrent tickets per user has been chosen as equal to 6.



**15.1 Number of tickets check and creation of a new ticket**

* **R16**: The system must allow the user to check their pending tickets → *Implemented*

From the main page, the user can view their underway tickets (the system reads them from the Account Database). Clicking on the "check" button on the right of each ticket, the user can access its corresponding "ticket page" where there is the QR code and the other info about the reservation.

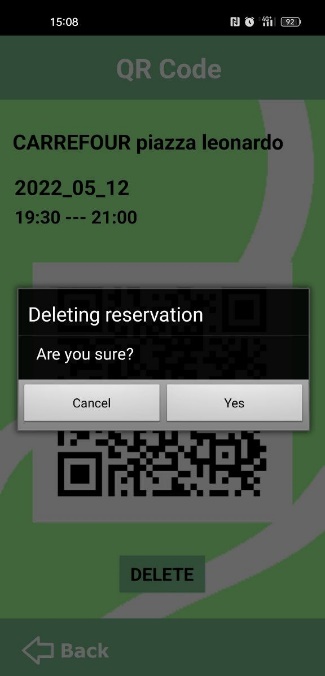
**16.1** **User's tickets on the main page 16.2 Ticket page after tapping on "CHECK"**

* **R17**: The system must allow the user to delete their pending tickets → *Implemented*

From the ticket page, the user can delete the ticket by pushing on the "delete" button: the system asks for confirmation and, in the positive case, the corresponding ticket is canceled from the Account Database.

Before deleting the ticket, the system reads the URL market attached to the ticket to access the corresponding "Market Database" and decrease the crowding in the time slots that were involved in the booking.

Note: at the moment the system just removes the ticket from the user's row. A possible future development is also to shift the other tickets when one is deleted in order to have them in chronological order.



**17.1 Ticket cancellation**



**17.2 Account Database before ticket cancellation**



**17.3 Account Database before ticket cancellation**

* **R18**: The system does not allow the user to change single settings of already generated tickets

→ *Implemented*

There are not buttons to change single settings of the booking in the ticket page. The user can only delete the current ticket and create a new reservation.

* **R19**: The system must automatically delete the ticket at the end of its reservation time

*→ Implemented*

The tickets are not instantaneously canceled from the database when they expire, but, each time the user accesses the Main Page, the system automatically acquires the current time from the device and compares it to the reservation time of the user tickets in the database in order to delete the ones that are expired.

Obviously, for each ticket deleted, the crowding in the corresponding market is not updated since, in this case, they were expired tickets.