POLITECNICO MILANO 1863

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

Alberto Valentini alberto.valentini@mail.polimi.it

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

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to present a possible way to implement a prototype of the mobile phone application “Customers 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 show how the application may be implemented for an Android phone, explaining the architecture realization and the programming language selected, and by analyzing the advantages and the downsides 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 CLup app 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.
- URL: Uniform Resource Locator.
- 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.
- 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 bookable 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.
- Apps Script web application: executable application via web of the JavaScript Google Apps Script code

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- DD: Design Document.
- TD: Testing Document.
- CLup: Customers Line-Up (name of the application).
- UserGUI: Graphical User Interface.

1.3.2 Abbreviations

- Rn: Requirement number n.

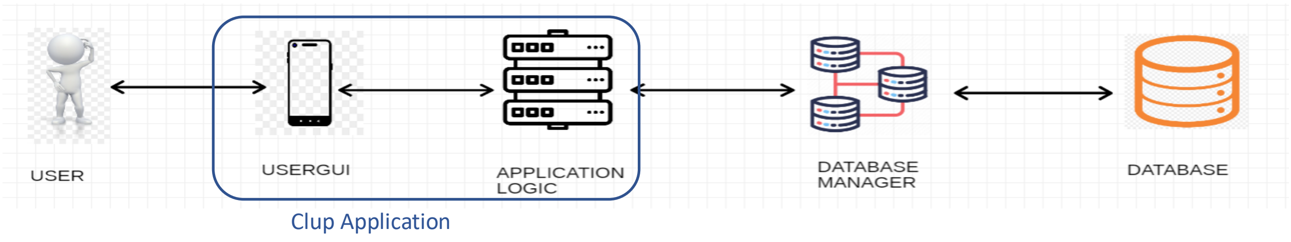
1.4 References

- Requirement Analysis and Specification Document – Customers Line-Up.
- Design Document – Customers Line-Up.
- Testing Document – Customers Line-Up.
- Project proposal: link to the document
- MIT App Inventor: App Inventor main page
- Google Sheets: Google sheet main page
- Google Apps Script: Google Apps Script main page
- GitHub repository: link to the project repository

2. System architecture

2.1 High level system overview

According to what is outlined in the Design Document, the system architecture is represented on a high level by the picture below. The user interacts with the UserGUI, then, the Application Logic provides the service by communicating with the Database thanks to the Database Manager.



High level system overview

In practice, the UserGUI and the Application logic are integrated into the unique system denoted as "CLup application", which is the actual app downloaded by the user.
The Application Logic represents the core which makes the application operational.

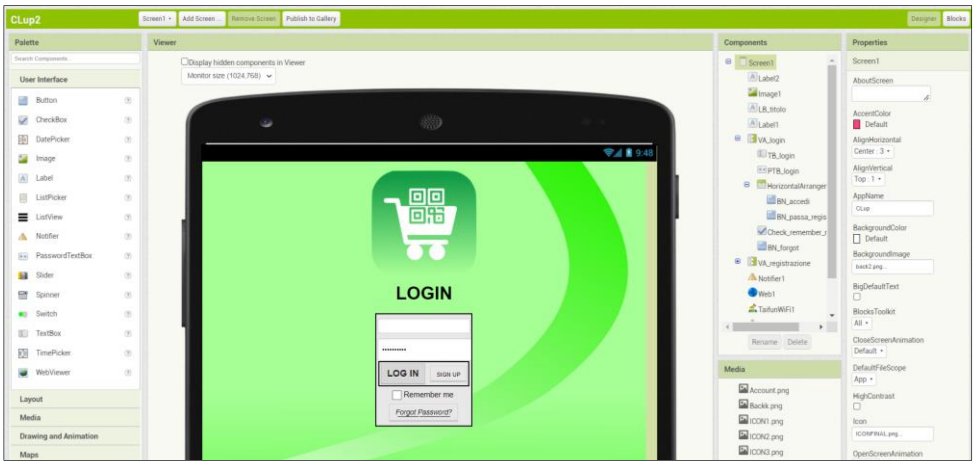
2.2 CLup Application

The CLup application is the fundamental element of the system. It processes the incoming request from the user and, when necessary, invokes the Database Manager to get/store data from/to the Database to fulfill the demand.

The CLup application has been developed for Android phones by taking advantage of the online framework "MIT App Inventor". App Inventor allows the creation of both Android and IOS applications in a quick and intuitive way owing to the availability of a wide range of ready-to-use objects and methods that the programmer just needs to code through a drag-and-drop approach.
For the current prototype, it has been opted for an Android version due to testing reasons (the developers are Android users) and because the IOS App Inventor environment is just recently introduced, hence potentially exposed to some bugs.

App Inventor organizes the application in "screens": each screen represents a stand-alone module of the app dedicated to a specific service. Therefore, each time is needed to pass from one screen to another, the programmer must take care of passing the right information in order to not lose the track of what the user was doing.
In the App Inventor environment, each screen is developed by using two sections:

- Designer: here the user interface is designed, deciding which components the app must have and setting the layout.



App Inventor Screen "Designer" view

- Blocks: this is the coding section. Here the programmer must set the components' behavior, the data processing and the interfaces for external communications (e.g. with the database manager).

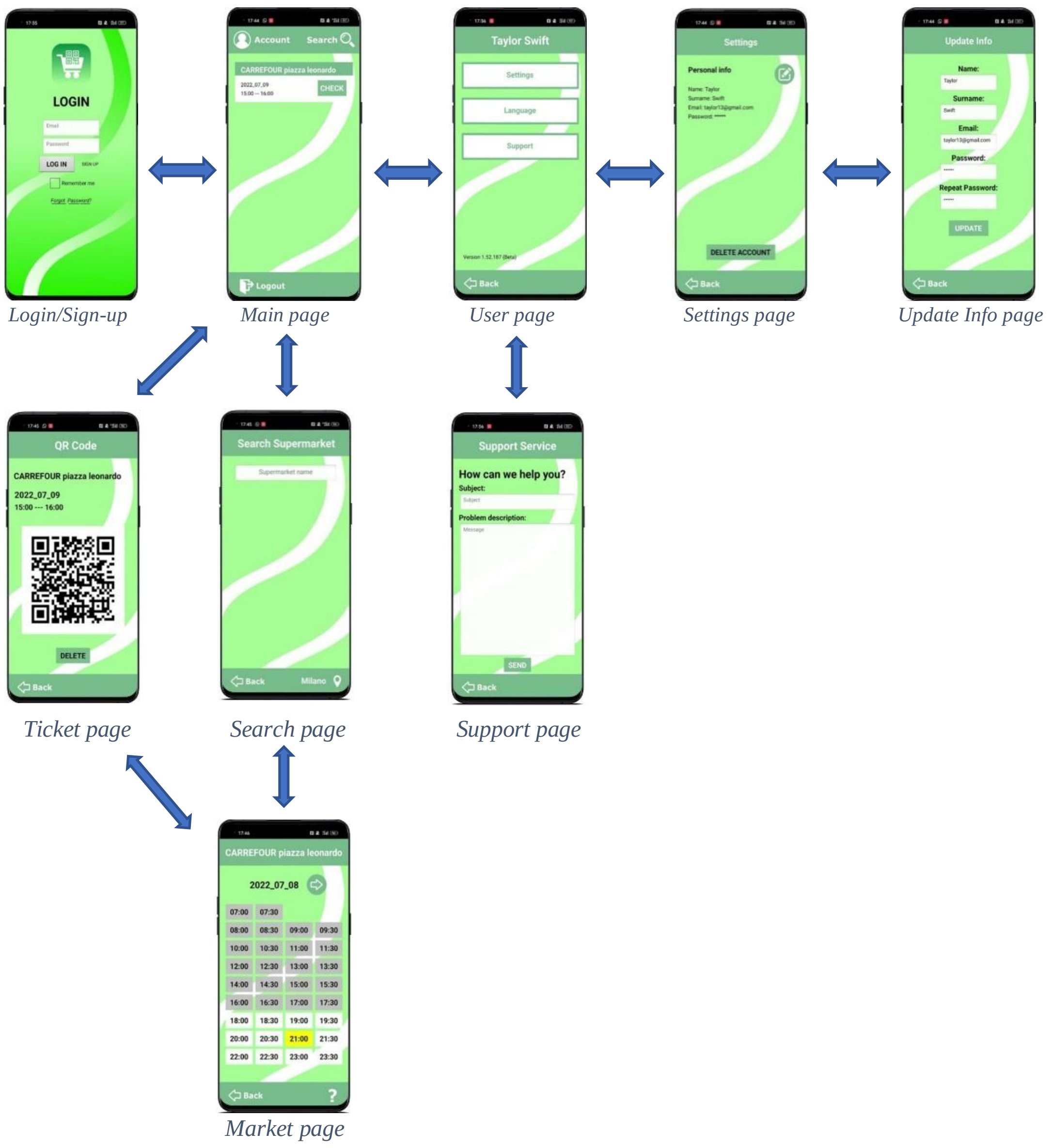
App Inventor Screen "Blocks" view

The implemented Clup application consists of 9 screens ("button" → *screen*):

1) *Login/Sign-up*: it allows new users to sign up and already registered users to log in.
   "Login" → *Main page*
2) *Main page*: it is the main page of the app, where the user can check their pending tickets or move to the other app's sections.
   "Account" → *User page*
   "Search" → *Search page*
   "Check" → *Ticket page*
   "Logout" → *Login/Sign-up*
3) *User page*: this is a transition page. Here the user can select other pages based on which action they want to do.
   "Settings" → *Settings page*
   "Support" → *Support page*
   "Back" → *Main page*
4) *Settings page*: here the user can see some settings related to their account. Then, they can decide to modify them or even delete the account.
   "Modify info" → *Update Info page*
   "Delete account" → *Login/Sign-up*

“Back” → *User page*

5) *Update Info page*: here the user can change some settings related to their account.
   “Update” / “Back” → *Settings page*
   “Back” → *Settings page*
6) *Support page*: from here the user can contact the app assistance.
   “Send” / “Back” → *User page*
7) *Search page*: here the user can select the city and look for the supermarket.
   “Market selection” → *Market page*
   “Back” → *Main page*
8) *Market page*: here the user can book a ticket in one of the available time slots.
   “Confirm reservation” → *Ticket page*
   “Back” → *Search page*
9) Ticket page: here the user can check the ticket with the QR code and delete it if they want.
   “Delete” → *Main Page* or *Market page* depending on the last screen
   “Back” → *Main Page* or *Market page* depending on the last screen



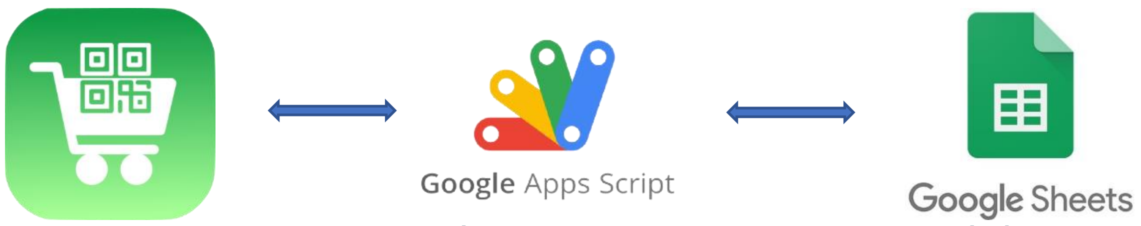
Further details on the app behavior will be given in the "Requirement implementation" section.

App inventor has the clear advantage of simplifying the app development to untrained programmers compared to the classic OO languages. However, this also results in a poor code readability and portability. Moreover, it lacks in flexibility, since the control structures are limited to the ones provided by the environment, it has the flaw of not supporting background activities and we have noticed during the implementation as the app is affected by relatively high response times. Therefore, App Inventor can be considered as useful tool to develop app prototype quickly, but the actual application should be carried out pursuing other ways.

2.3 Database management

A database is a systematic collection of data. It contains the info needed for the proper working of the application. The application does not directly access the database but passes through the Database Manager System. The DBMS encompasses methods to manage the data in the Database and fulfill the client demand, such as returning data read from the DB or storing data in it. In this way, the information in the Database remains hidden from the application and its working does not directly depend on the way they are implemented in the DB.

As a database, in our prototype, we have used Google Sheets. They are free online spreadsheets and they have been chosen for their ease to use in combination with App Inventor and Google Apps Scripts. Google Apps Script is basically JavaScript code, but with a wide library of functions to work with Google Sheets. In our system, they have the role of the Database Manager System.



CLup app *Google Apps Script as DBMS* *Google Sheets as DB*

In our implementation, we did not use just a single Google Sheet as DB, but several Google Sheets depending on the nature of the data stored, so in a structure that resembles a distributed database; this way, it results easier handling a great amount of data and it improves the maintainability. Each Google Sheet has its relative Google Apps Script for reading the data, instead, there is only one Google Apps Script for all the Google Sheets for writing data, since, generally, a storing operation involves more Google Sheets. The Google Apps Scripts are invoked by the CLup application using the App Inventor component "Web", which establishes an internet connection to execute the web application of the Google Apps Script. The Google Apps Scripts return the database content as a JSON object. Then, the CLup application decodes it and takes only the information needed for that specific function.
Below the links for the Google Sheets and Google Apps Scripts used.
For the sake of simplicity, in the prototype just two cities (Milan and Rome) and two supermarkets per city have been implemented, but the logic is the same and easily extendable for more cities and markets.
Note: to access the Google Apps Script a Google account is requested

1) Account DB: account info and ticket storing
   - Google Sheet
   - Google Apps Script: data reading

2) City DB: supported cities info
   - Google Sheet
   - Google Apps Script: data reading

3) Milano DB: info of the supported supermarkets in Milan
Google Sheet
Google Apps Script: data reading

4) Roma DB: info of the supported supermarkets in Rome
Google Sheet
Google Apps Script: data reading

5) Esselunga Rubattino DB: info on the given supermarket (Milan)
Google Sheet
Google Apps Script: data reading

6) Carrefour Leonardo DB: info on the given supermarket (Milan)
Google Sheet
Google Apps Script: data reading

7) Coop Colosseo DB: info on the given supermarket (Rome)
Google Sheet
Google Apps Script: data reading

8) Famila Rebibbia DB: info on the given supermarket (Rome)
Google Sheet
Google Apps Script: data reading

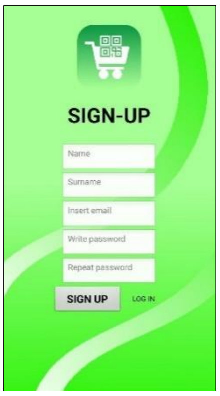
9) Google Apps Script for storing data

Also in this case, the proposed solution can be acceptable for a prototype, but for the actual application is suggested the switch to more robust database structures. In fact, the combination Google Sheet - Google Apps Script can be fine for a limited number of users, but managing a high number of users and concurrent requests may become critical.

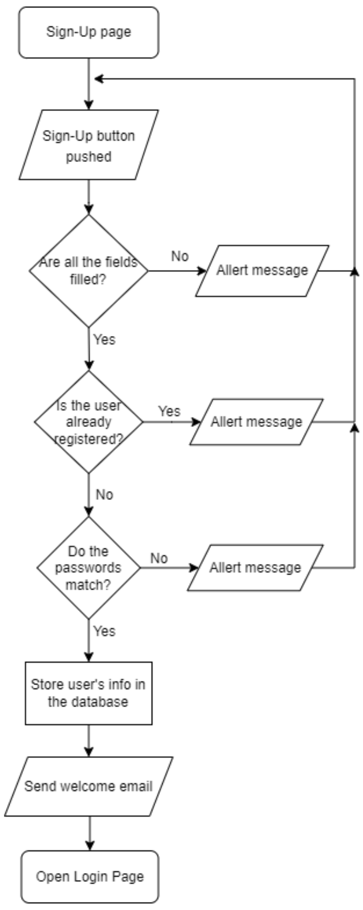
3. Requirements implementation

According to the section 3.2 of the Requirements Analysis and Specification Document, below the requirements of the application are listed, specifying if they have been implemented or not, and, in the positive case, a brief description of the working principle is given, showing the integration and connection between the previous described modules.

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

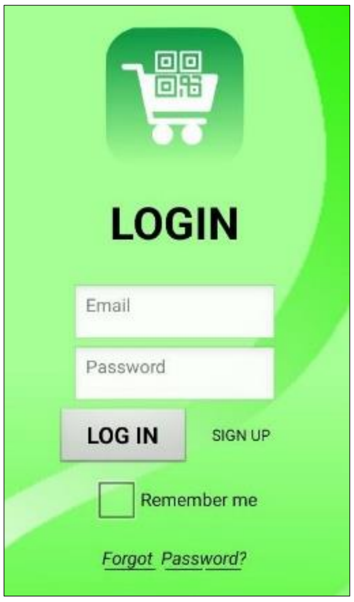


1.1 Sign-up page

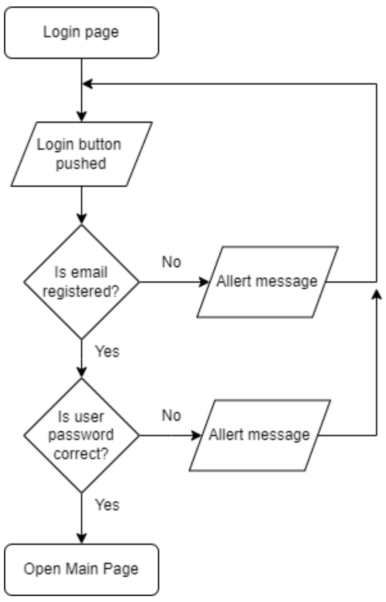


1.2 Working Principle

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

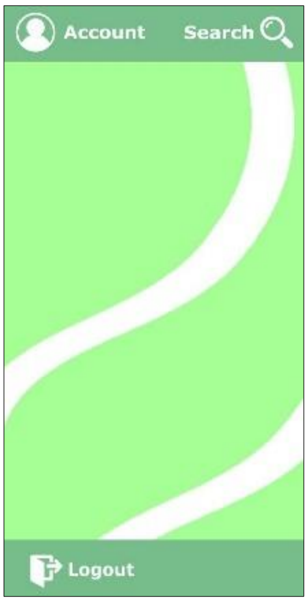


2.1 Login page

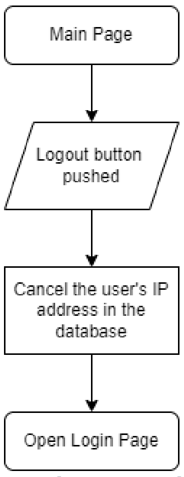


2.2 Working Principle

- **R3**: The system must allow the user to log out → *Implemented*
  Note: see requirement R21 for further info about IP address management

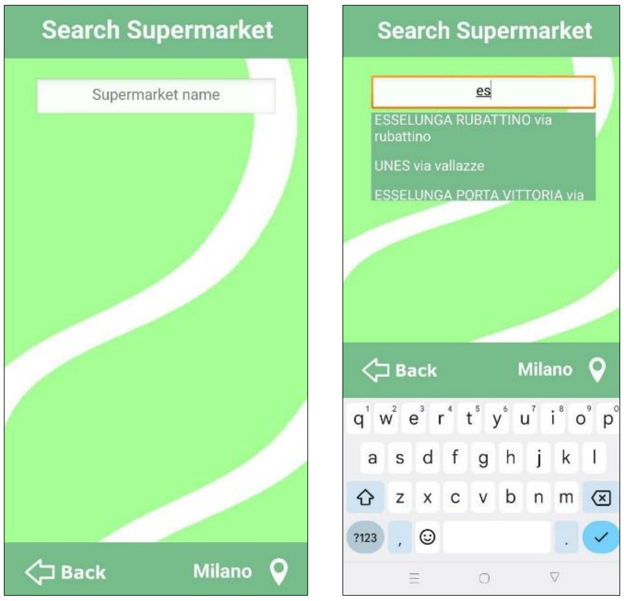


3.1 Main Page



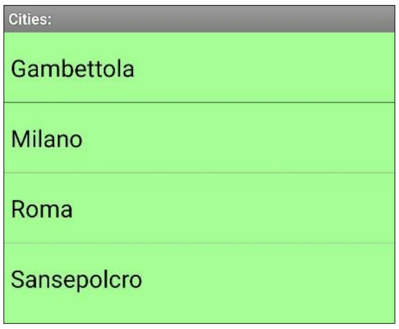
3.2 Working Principle

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

- **R5**: The system must allow the user to choose the city → *Implemented*
  The default user 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 applied on the *Search page* and registered in the Account DB for future times.

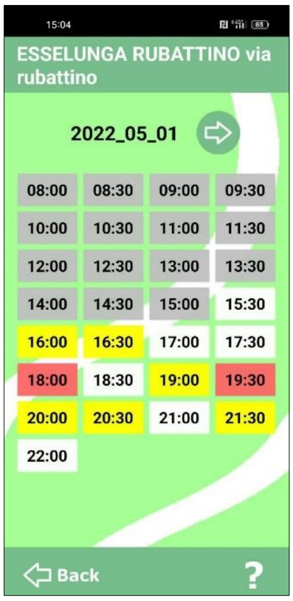


5. City Selection

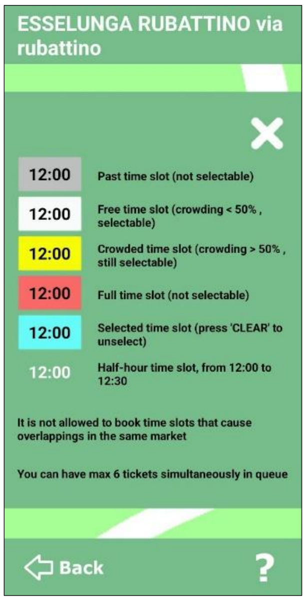
- **R6**: The system must restrict the supermarket research to the ones that are in the user city → *Implemented*
  1) Read the user city from Account DB.
  2) Read the corresponding URL of the apps script web application from City DB
  3) Use the URL to read the data from the database (e.g. Milano DB or Roma DB) where are listed the city's supermarkets and the research is restricted to only these ones.

- **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 relative apps script web application URL is got
2) The URL is used to access the market database (e.g. Esselunga Rubattino DB).
3) In the user interface, the time slots are represented as buttons with impressed the slot start time, so there can 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 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 current time, are marked in gray.
   The crowding levels of the day are registered in the DB. If the booking number of a time slot is equal to the maximum capacity set by the store manager, 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, 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.



7.1 Market timetable



12:00	Past time slot (not selectable)
12:00	Free time slot (crowding < 50% , selectable)
12:00	Crowded time slot (crowding > 50% , still selectable)
12:00	Full time slot (not selectable)
12:00	Selected time slot (press 'CLEAR' to unselect)
12:00	Half-hour time slot, from 12:00 to 12:30

It is not allowed to book time slots that cause overlappings in the same market

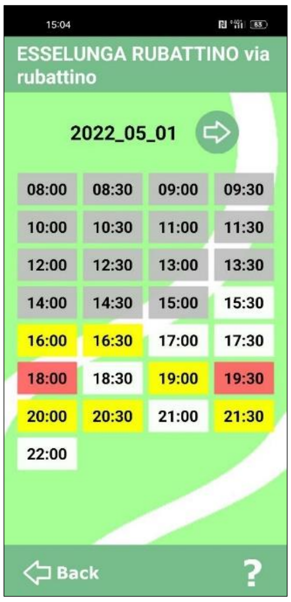
You can have max 6 tickets simultaneously in queue

7.2 Legend

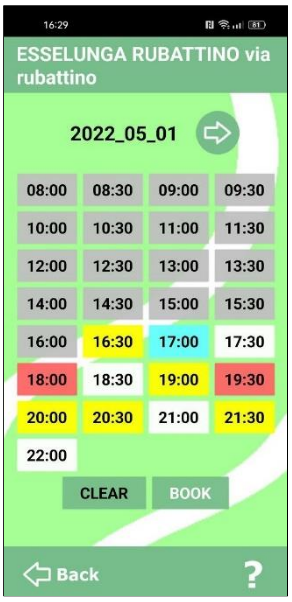
- **R8**: The system must allow the user to select the day on which they want to book in the selected supermarket, among the ones that the store manager makes available → *Implemented*

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.
Then, every time the booking day is changed, the system does the same operations as described in R7.3, 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.

- **R9**: The system allows the user to make a reservation in only 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.

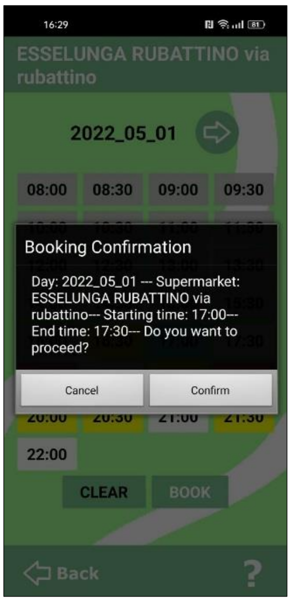


9.1 Anything selected



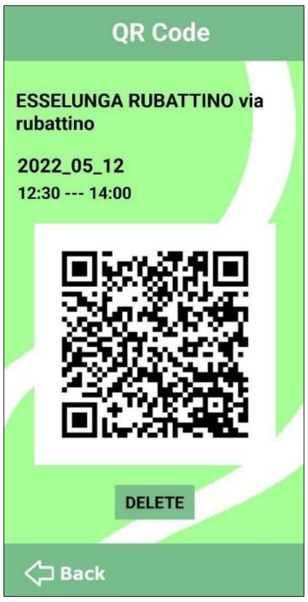
9.2 Free time slot button selected

- **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.


Booking Confirmation

Day: 2022_05_01 --- Supermarket: ESSELUNGA RUBATTINO via rubattino--- Starting time: 17:00--- End time: 17:30--- Do you want to proceed?

10.1 Booking Confirmation

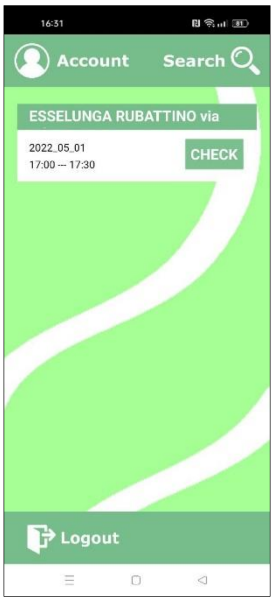

QR Code

ESSELUNGA RUBATTINO via rubattino

2022_05_12

12:30 --- 14:00

10.2 Ticket page


ESSELUNGA RUBATTINO via

2022_05_01
17:00 --- 17:30

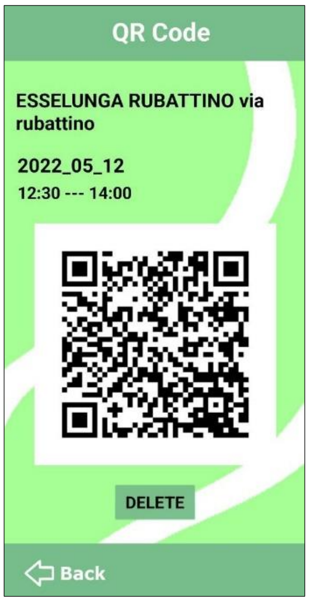
10.3 Ticket in Main page

"Cancel" brings the user back. if there are no incompatibilities with already existing user tickets (see R14 and R15 for exceptions), clicking on "Confirm", instead, the ticket is created.
In Account DB, the following booking info are stored: market name, reservation day, start and end time of the booked time slot, market database URL. This URL will be then used by the Google Apps Script to update crowding data when the ticket is deleted.

- **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, then, the increment is made for each of the time slots in the booking range.
  On the other hand, when a user ticket is deleted, the system uses the URL stored during ticket to access the database of the booked market and 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*
  Since the ticket control is delegated to the supermarket personnel, the info contained in the QR should be decided by mutual agreement. In our prototype, the QR code contains the user email, the booked market name, and the day and time of the reservation.

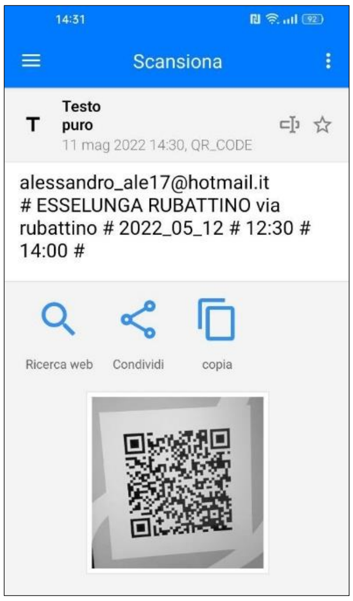

QR Code

ESSELUNGA RUBATTINO via rubattino

2022_05_12

12:30 --- 14:00

12.1 Ticket page



alessandro_ale17@hotmail.it # ESSELUNGA RUBATTINO via rubattino # 2022_05_12 # 12:30 # 14:00 #

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, the Google Apps Script checks for overlapping issues with each of the eventual already existing user tickets at the same market and on the same day.

Denoting as *request-start*, *request-end*, *existing-start* and *existing-end* the start and end time of the requested and already existing ticket at the same market on the same day, the script controls if one of the following conditions occur:

- $request\_start \geq existing\_start \ \ AND \ \ request\_start \ < existing\_end$
- $requested\_end \geq existing\_start \ AND \ requested\_end \leq existing\_end$
- $requested\_start \leq existing\_start \ AND \ requested\_end \geq existing\_end$

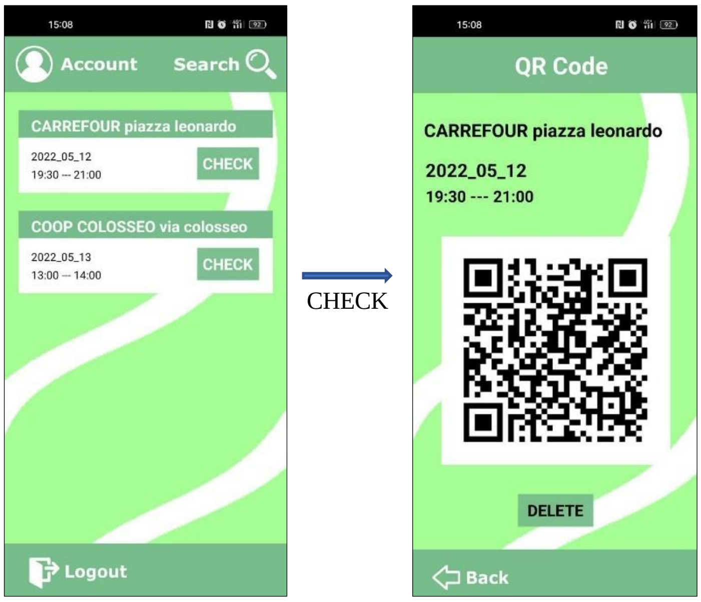
If one of the above conditions is true, then the system returns an overlapping error.

- **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 set equal to 6.

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

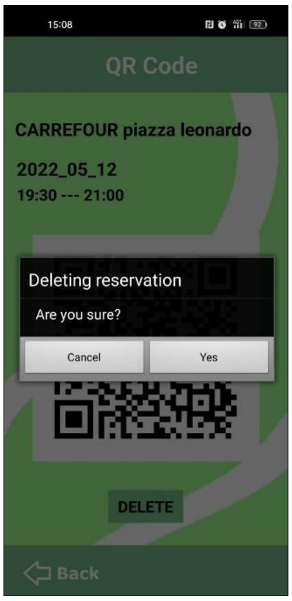
  From the *Main page*, the user can view their underway tickets. Clicking on the "check" button on the right of each ticket, the user can access its corresponding *Ticket page*, where there is the QR code and the other info about the reservation.



16.1 Main page *16.2 Ticket page*

- **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.
  Before deleting the ticket, the system reads the URL market attached to the ticket in the Account DB to access the corresponding market database and decrease the crowding in the time slots that were involved in the booking.



CARREFOUR piazza leonardo

2022_05_12

19:30 --- 21:00

Deleting reservation

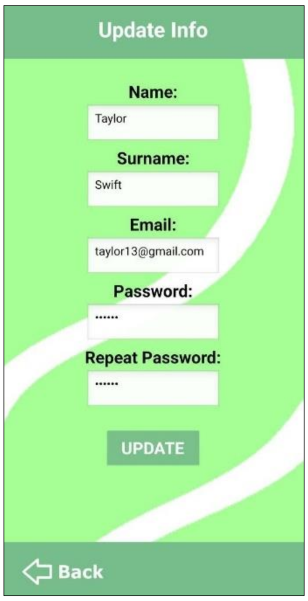
Are you sure?

17.1 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 one.

- **R19**: The system must automatically delete the ticket at the end of its reservation time
  → *Implemented*
  The tickets are not instantaneously canceled when they expire, but, each time the user accesses the *Main page*, the system automatically acquires the current time from the user device and compares it to the tickets' reservation time in order to delete the ones that are expired.
  Apparently, for each ticket deleted, the crowding in the corresponding market is not updated since, in that case, they were expired tickets.

- **R20**: The system should allow the user to update their data and delete their account → *Implemented*
  From the Update Info page, the user is able to modify some settings related to their account. Clicking on the button "Update" the changes are registered to the Account DB.



Name: Taylor

Surname: Swift

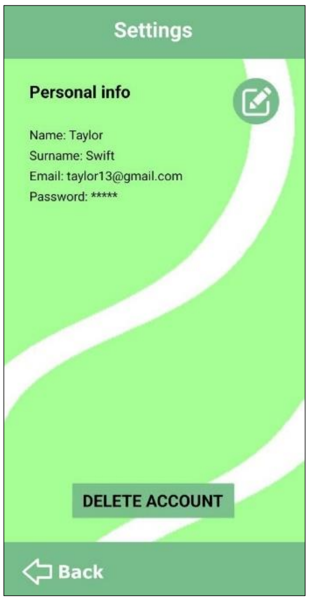
Email: taylor13@gmail.com

Password: ......

Repeat Password: ......

20.1 Update Info page

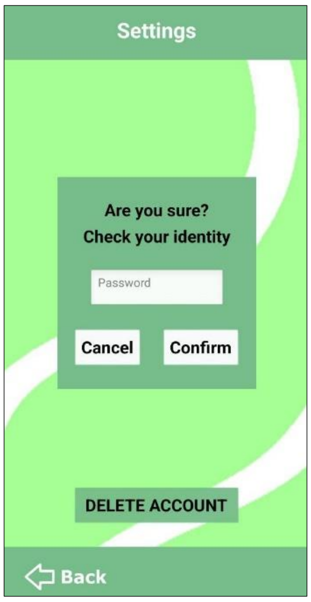
Instead, if the user wants to delete their account, from the *Settings page* they have to push on the "Delete Account" button.



Name: Taylor
Surname: Swift
Email: taylor13@gmail.com
Password: *****

20.2 Settings page

Delete Account

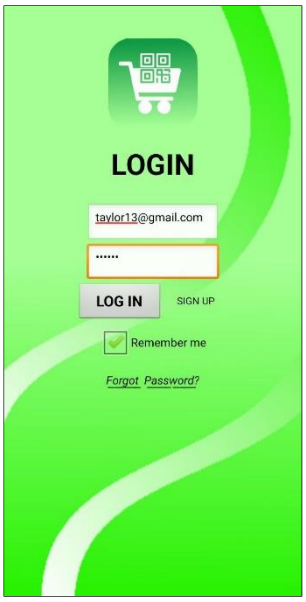


20.3 Delete Account

Once the user has inserted their password and selected “Confirm”, the account is deleted.

- **R21**: The system should allow the user to auto-log in if they have previously ticked a “Remember me” box → *Implemented*

If during the login phase the user ticks the "Remember me" box, the system stores the user's IP address in the Account Database, so that next times the user does not need to insert their credentials, but the system automatically logs it in.
Obviously, this is still true until the user does not change their internet connection, otherwise, they need to do the login again.

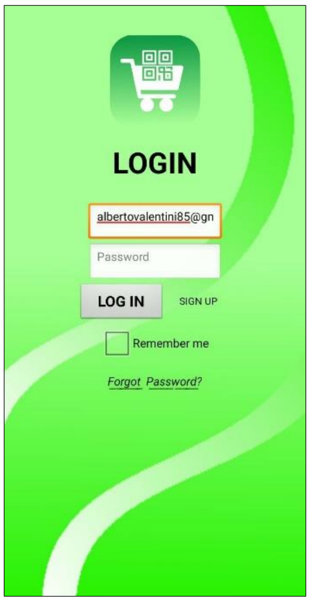


taylor13@gmail.com
••••••

21. Remember me - auto login

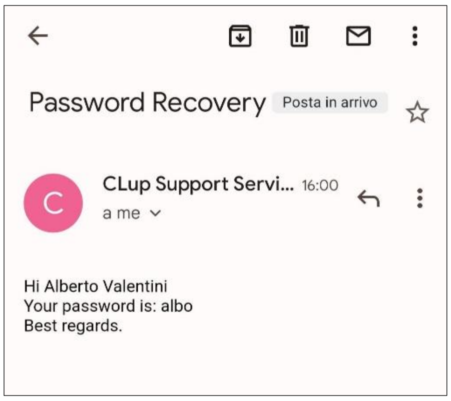
- **R22**: The system should allow the user to recover their credentials → *Implemented*

  If the user forgets their password, from the *Login page* they can push the "Forgot Password?" button. The system sends an e-mail to the user's email address containing their password.



22.1 Login page

Forget password?


Password Recovery Posta in arrivo

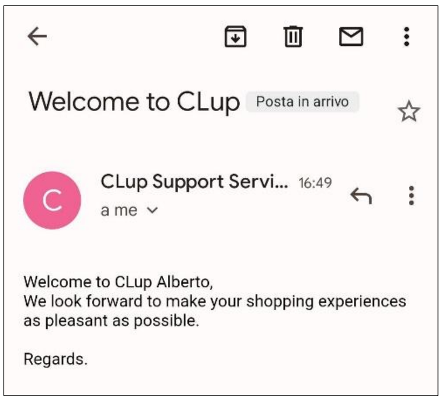
CLup Support Servi... 16:00
a me

Hi Alberto Valentini
Your password is: albo
Best regards.

22.2 Recovery password e-mail

- **R23**: The system should alert the user when their account is created/deleted → *Implemented*

  When a new user signs up, the system sends a welcome e-mail to their email address.
  In the same way, when the user deletes their account, the system sends an e-mail to their email address to warn them of the operation.

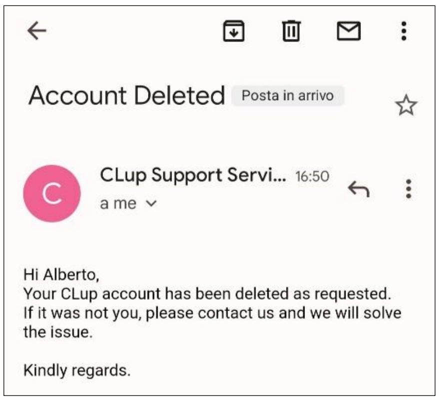

Welcome to CLup Posta in arrivo

CLup Support Servi... 16:49
a me

Welcome to CLup Alberto,
We look forward to make your shopping experiences as pleasant as possible.

Regards.

23.1 Welcome e-mail


Account Deleted Posta in arrivo

CLup Support Servi... 16:50
a me

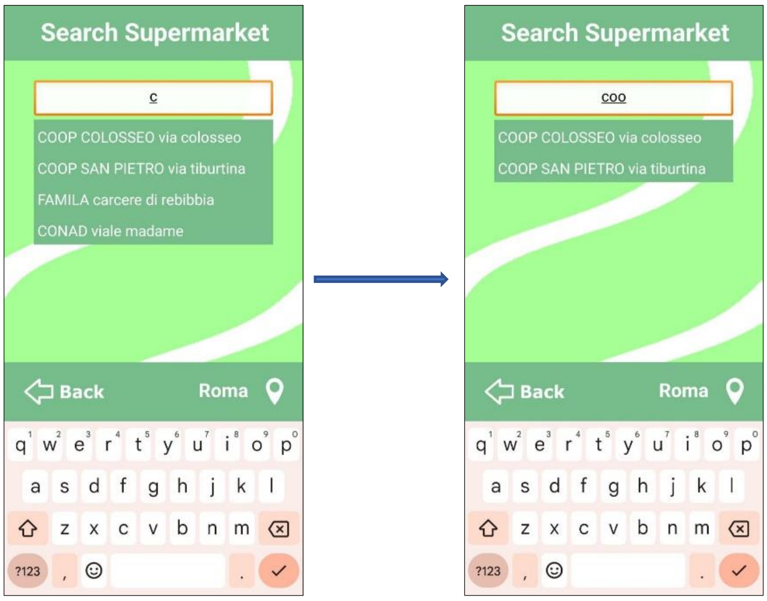
Hi Alberto,
Your CLup account has been deleted as requested.
If it was not you, please contact us and we will solve the issue.

Kindly regards.

23.2 "Account deleted" e-mail

- **R24**: The system should filter supermarkets by name → *Implemented*

  As already explained in requirement R4, in the *Search* page, the supermarkets are first filtered by user city. Then, the research among the supermarkets in the city is carried out by market's name. Below an example:



c

COOP COLOSSEO via colosseo
COOP SAN PIETRO via tiburtina
FAMILA carcere di rebibbia
CONAD viale madame

coo

COOP COLOSSEO via colosseo
COOP SAN PIETRO via tiburtina

24. Search market by name

- **R25**: The system should remind the user about incoming reservations before their beginning → *Not Implemented*

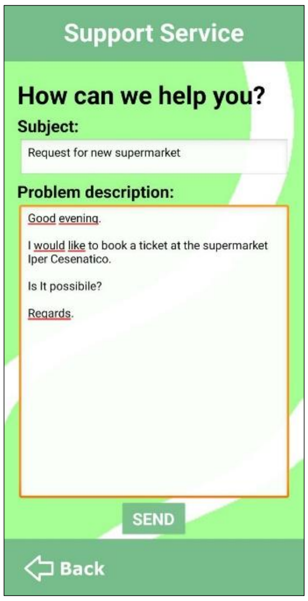
  This requirement has not been implemented because App Inventor does not support applications with background activities.

- **R26**: The system should allow the user to select the app language → *Not Implemented*

  This requirement has not been implemented for matter of time since it is just a "good to have" requirement. It might be a future development.

- **R27**: The system should allow the user to contact the assistance → *Implemented*

  In the *Support page*, the user can report an error or issue, then the CLup development team will receive an e-mail with the specified problem description.



How can we help you?

Subject:

Request for new supermarket

Problem description:

Good evening.

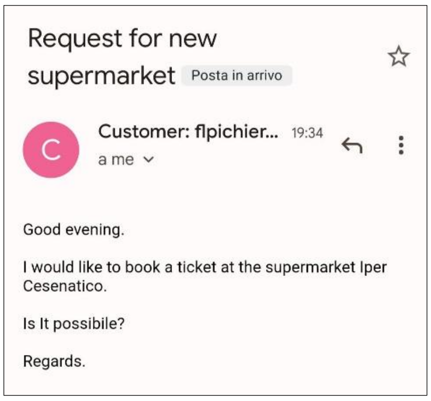
I would like to book a ticket at the supermarket Iper Cesenatico.

Is It possibile?

Regards.

27.1 User request

SEND



Request for new supermarket

Customer: flpichier... 19:34

Good evening.

I would like to book a ticket at the supermarket Iper Cesenatico.

Is It possibile?

Regards.

27.2 E-mail to the developers

- **R28**: The system should allow the user to download the QR ticket to use it offline → *Not Implemented*

  For the sake of simplicity, this "good-to-have" requirement has not been implemented. However, in case of necessity, the user can take a screenshot of the QR code.

- **R29**: The system should show the recent supermarket searches → *Not Implemented*

  This requirement has not been implemented for matter of time since it is just a "good to have" requirement. It might be a future development.

- **R30**: The system should memorize the favorite supermarkets → *Not Implemented*

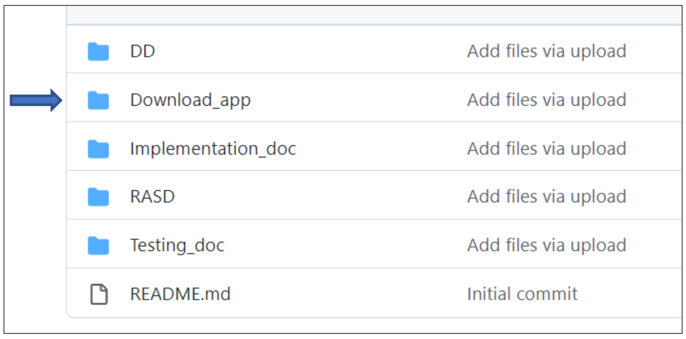
  This requirement has not been implemented for matter of time since it is just a "good to have" requirement. It might be a future development.

4. Installation guide

There are basically two ways to install the application on an Android smartphone:

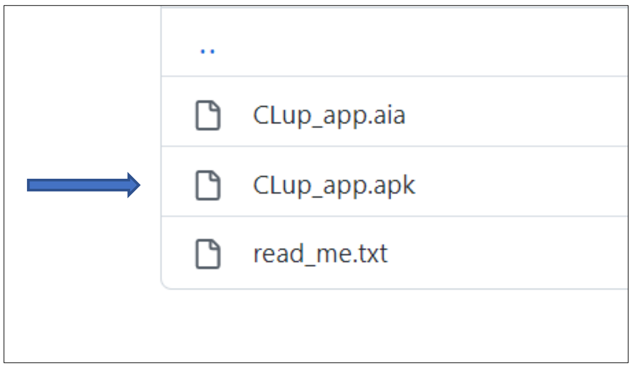
1) Download the "CLup_app.apk" file

   The user can download the apk file from the GitHub repository and, then, they just need to execute it once transferred to their smartphone.



DD	Add files via upload
Download_app	Add files via upload
Implementation_doc	Add files via upload
RASD	Add files via upload
Testing_doc	Add files via upload
README.md	Initial commit

GitHub project repository

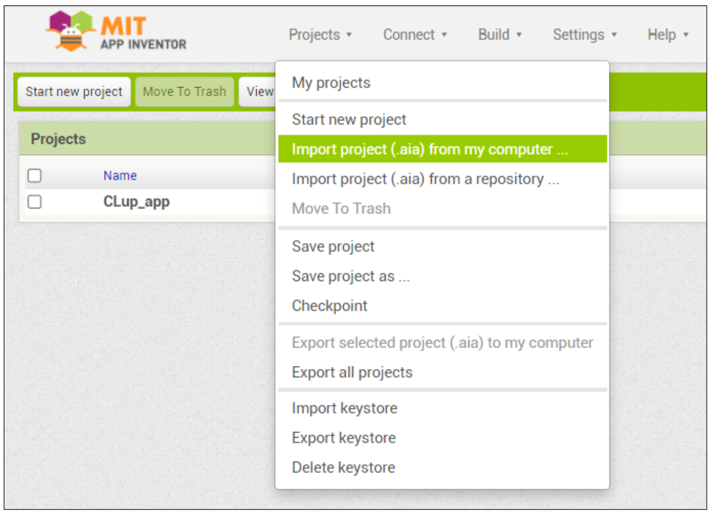


..
CLup_app.aia
CLup_app.apk
read_me.txt

Download CLup_app.apk

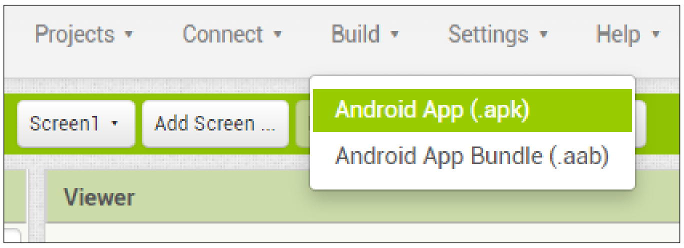
The user installing the application may need to change the settings of their phone to allow the installation of non-market applications.

2) The second method is less straightforward. Instead of the apk, the user can download from GitHub the "CLup_app.aia." file. It is the actual App Inventor project and it needs to be imported into MIT App Inventor. Once the user has made access App Inventor with their Google account, they need to import the .aia file:



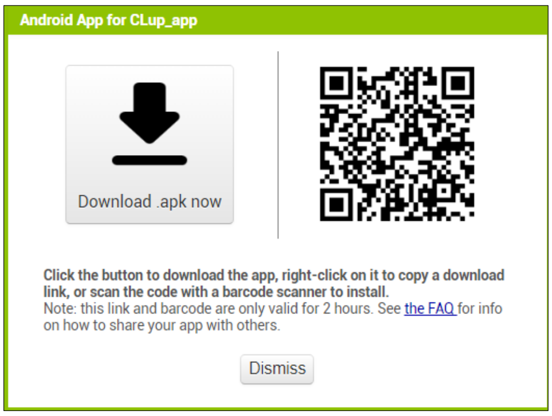
Import the project into App Inventor

Next, the user just needs to open the project, click on the "Build" menu from the App Inventor toolbar and, finally, on the "Android App (.apk)" item



Build -> Android app (.apk)

App Inventor starts to build the app. When the compilation is finished, the user can download the .apk file and proceed as described before, or simply scan the given QR code with the smartphone (beware: the QR code is valid for only two hours)



Click the button to download the app, right-click on it to copy a download link, or scan the code with a barcode scanner to install.
Note: this link and barcode are only valid for 2 hours. See the FAQ for info on how to share your app with others.

Scanning the QR code, the apk download starts on the phone automatically. Then, the user just needs to open the apk file to install the CLup application.