Mobile app development report: CShare

Clara Gros
Babacar Toure

February 2020

Abstract

As part of our second year project at CentraleSupélec in the science media pole, we chose to focus on the development of a food redistribution application to fight against food waste, intended for the scale of a university residence. This application would allow students to donate their surplus food and collect others' food for free. That aims at limiting the ecological impact of students on a campus.

Contents

1 Introduction

As part of our second year project at CentraleSupélec in the science media pole, we chose to focus on the development of a food redistribution application to fight against food waste, intended for the scale of a university residence. This application would allow students to donate their surplus food and collect others' food for free. That aims at limiting the ecological impact of students on a campus.
This project is scheduled to last at least an entire year and has already allowed us to acquire several skills in software development from conception to design of the software. Indeed, the development of a mobile application includes the client and the server side which both requires different programming skills. We were therefore lead to explore a wide range of techniques as well as technologies to conduct this project.
This partial report compiles all the work we have done so far (up to March). It therefore contains the documents that allowed us not only to define the need and the product, but also to define our development strategy in terms of the technologies used, the user interface and the architecture. Until then, we are able to provide a first functional but not optimal MVP (Minimal Viable Product) so that we can test it with a panel of customers, retrieve their opinions and thus start the development of a finished and optimal product.

2 Specifications

2.1 Stakes : Identification of needs

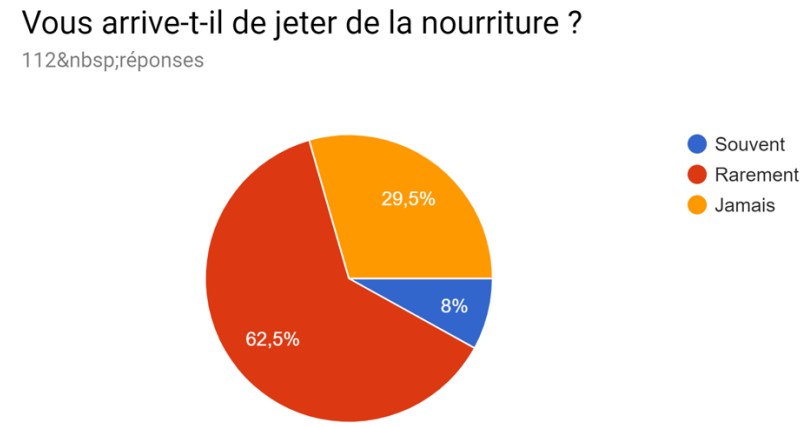
2.1.1 General concept

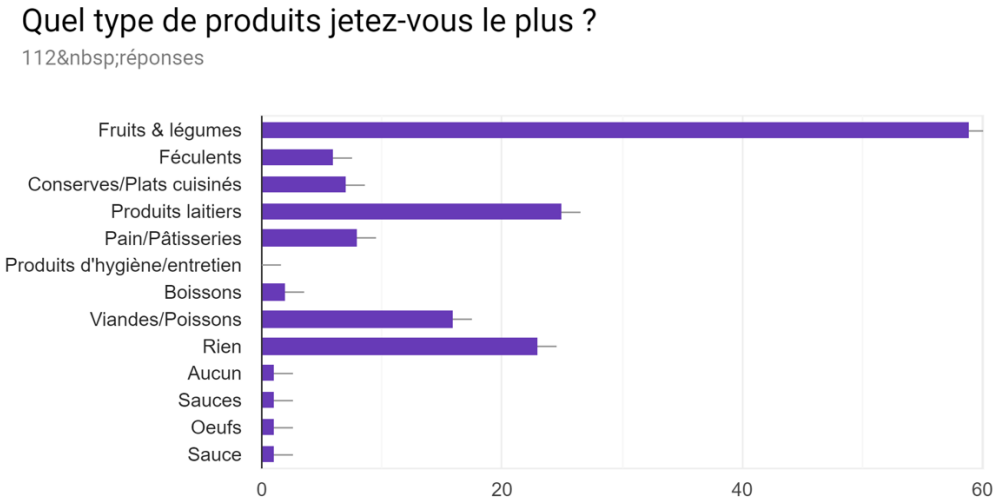
Our goal is to develop an app that allows users who are residents on a university campus to be able to give away their excess food and collect others' for free. This project was inspired by the other similar already existing app called TooGoodToGo: aware of the extent of food wastage among university students, we decided to develop an app that would allow for redistribution of unused food on the scale of a campus.

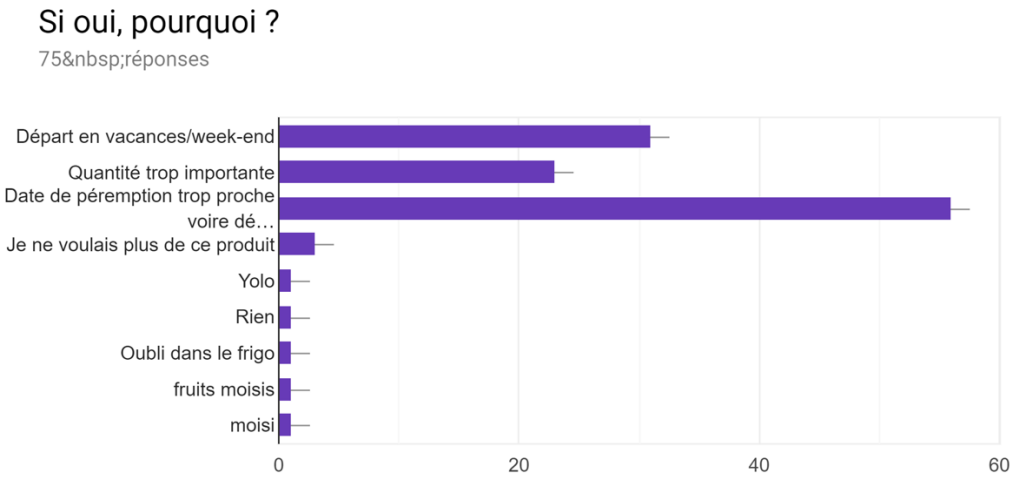
2.1.2 Market study

We first wanted to estimate how receptive university students would be: for that, we carried out a survey (see Appendix A) and successfully reached out to over a hundred students from several campuses: the goal of this survey was to evaluate and refine the need for such an application, as well as quantify the potential user base we could expect to have.

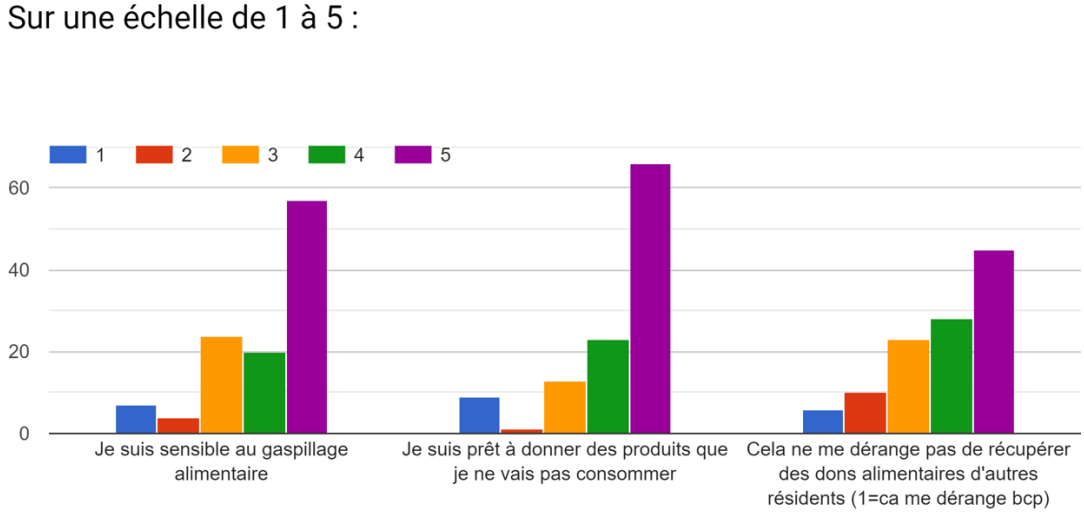
Need One of the objectives of this survey was to evaluate how much food is wasted and the reasons for that wastage: what we found out reported in the charts below is that over two thirds of interrogated students said to waste food products, the main ones being fruits, vegetables and dairy products: the reason cited by three quarters of respondents was products were closing in on or had already passed their expiration date, with another major cause being leaving campus on holidays.







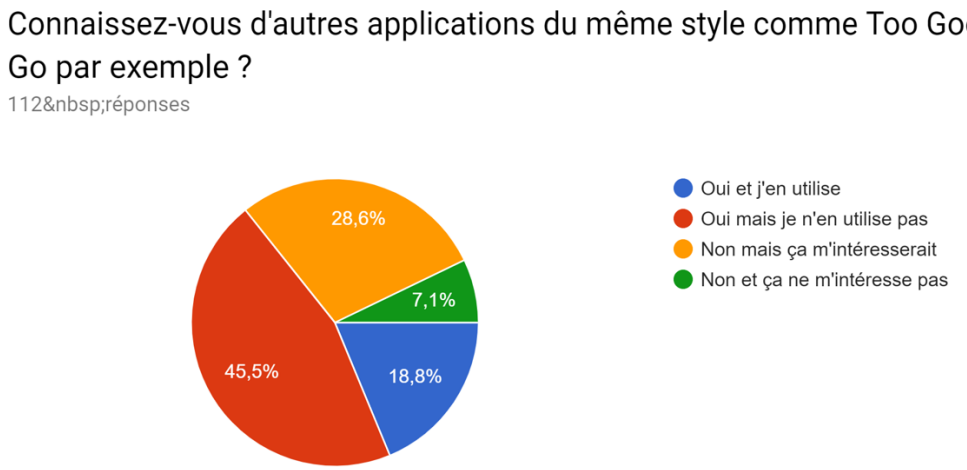
Limitations We were also curious of the possible limitations and drawbacks this project would have, and according to respondents one limitation is users' willingness to accept food products from other people, which as seen in the chart below is less on average than their willingness to give away unwanted products, although not by as much as we would have thought.



Some also explicitly mentioned the safety risk of food coming from an unverified source:
"Ce qui me dérangerait, c'est de ne pas savoir comment a été stockée la nourriture et de ne pas savoir par exemple si la chaîne du froid a bien été respectée".

That's why we thought of the possibility to later add a 5 stars grading system to ensure the supplier good faith' to give good products or at least allow users of the application to be able to report any malicious user.

Already existing apps There already exist similar applications that do not however target a geographically localized customer base like we plan to do. We asked the students if they knew of one and in that case if they actively used one, or if they didn't know of one and in that case were interested in using one. We were pleasantly surprised to discover the amount of respondents (over a quarter) who didn't know of any such app but were willing to start using one:



From this chart we can see that less than a third of people who knew of such apps use one, but close to 80% of respondents who didn't were interested in using one, from which we can deduce currently available apps do not fulfill the demand criteria well enough and/or do not appeal enough, an issue we hope to overcome with our app.

Support We have received very positive reviews on our project, and many verbally expressed their support:
"Une excellente initiative !"
"Très bonne idée d'appli !"

2.2 Scope

2.2.1 Objectives

Our main objective is to code an application along with a Web app for admin and a database. The application is expected to allow users to create an account and specify their personal information such as name and room number, and be able to log in and out: they then have access publications corresponding to food product offers and are able to accept them and create their own: they should then be put in contact with whoever user published the offer so that they can come to an agreement over the transfer of the product. Users may withdraw their own offers at any time and have the option to report offers and messages that do not fit Terms of Use (see 2.2.4 Safety constraints section below).

2.2.2 Resources and constraints

Time constraint: We are limited on time as the project is expected to only last until the end of the school year: further to that, we have a mid project presentation in January which correspond to the early departure of one team member.

Financial constraint: We have no budget as in the currently expected state of the project by the end of the year we have no planned expanses (see 4.4 Development prospects section).

Human constraint: We have little experience in application development, which is why the beginning of the project is reserved for learning using provided tutorials before actually starting the programming phase.

2.2.3 Already existing solutions

As mentioned earlier, other applications exist that fulfill a similar role, the most widely used one being Too Good To Go. However this application functions differently, as it targets grocery stores willing to sell their excess food at a lower price to individuals, thus having to separate customer bases. We found others like HopHopFood that offer a product similar to ours, but they are deployed on a much larger scale with no real geographical limitation which makes it unpractical. We have decided to instead aim at a more restricted but potentially more loyal user base.

2.2.4 Safety constraints

As we plan on maintaining databases that can contain personal information regarding users, it is necessary to tackle information security. As our application will offer open communication among users it is also necessary to impose adequate Terms of Use as well as allow active moderation of publications (see 4.4 Development prospects section).

3 Implementation

3.1 Tools and development environment

3.1.1 State of the art

Before developing the application, we scanned all existing technologies to make an informed choice regarding our choice of languages used but also software, frameworks and databases (see Appendix B.1).

3.1.2 Technical environment

In this section, based on the perimeter we defined and on the state of the art we wrote before, we are going to choose the best suited technologies to build our app. We will use two different technologies on the admin side and on the client side as well as one remote database.

Client-side The customer will have a mobile application and must be connected to the Internet to be able to make donations or collect food products. Given our time constraints, it will be a native mobile application because we don't have time to learn a new language neither to develop a hybrid application, nor to develop separately on Android and iOS. The choice therefore remains to be made between these two operating systems.With a combined market share of Android and iOS of about 99%, these systems have become unstoppable. However, Android can count on a market share four to five times higher than that of Apple. In Germany, for example, Android's market share is 81.5%, compared to 17% for Apple. The situation is similar on the Spanish and French markets. So we choose Android not only because the underlying programming language is Java (which we already know a little bit about) but also because on the scale of a university residence, we would like to reach as many people as possible and therefore develop on the most widespread OS. Still, we want to keep in mind that it remains possible to extend the project or to entrust it to future students in order to be able to develop the application under iOS (see 4.4 Development prospects section).

Server-side The role of the administrator would be more of a moderator role with regard to both users and food products circulating on the application. That's why we immediately thought of Django, a framework delivered by Python that almost instantly creates an administrator interface to keep control over the database with the basic CRUD operations. Thus, we would provide the administrator with a very simplistic Web app to moderate the mobile app. In addition, it will provide an API to communicate with the database without writing SQL queries but with a model system. Concerning the server itself, we will use an undetermined server (see with the campuses'network association). In the development phase we will use our computer as a local server.

Choice of the database For the choice of the server-side database, we chose a relational database. These are most popularly used and useful for handling structured data that organizes elements of data and standardizes how they relate to one another and to different properties. This is exactly what we need because we will have several tables (see Appendix B.2) and we will need to establish links between them all the time.
We have chosen to use the MySQL database manager. First of all, for economic reasons: we conduct a project using OpenSource products or products under free license from manufacturers. Certainly, if we were in a large project, we would have chosen a proprietary database option like Oracle (which is for many the best choice for study, practice or consensus). In addition, MySQL is quite intuitive and easy to use, even if these features are limited enough, it will be more than enough to meet the needs of our application.

Graphic design The graphic charter will be provided by us. Once it has been designed and written, it will be submitted to a panel of users to judge its relevance. It will group and codify:

- The logo that must be able to be adapted and declined on all supports. It will be found on flyers, websites and on Google Play store, for example.
- The fonts and typographical attributes to be used, generally a title font and a content font.
- The choice of colours through colour schemes adapted to the requirements of the different communication media and the different backgrounds available (coloured or white background). It must include either the Pantone, CMYK or RGB and hexadecimal values of each colour used.
- The rules for inserting each element (logo, title, baseline,...) and for each medium: margins and positioning in the document. Editorial rules (tone

and style) to help any writer to make his documents consistent.

To do so, we have planned to rely on the association CSDesign, which helps associations and clubs to create posters or logos. However, as we are neither an association nor a club, CSDesign charges us for its services and we have not yet found any other external contact to help us realize the graphic charter. This will be an issue we will think about once the development phase is over. Nevertheless we have a backup solution in case we don't find any external help: we are thinking of using Material.io (https://material.io/) which is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material.io streamlines collaboration between designers and developers, and helps teams quickly build beautiful products.

3.1.3 Class diagram

NOT ABLE TO PROVIDE THESE DOCUMENTS FOR NOW

3.2 Modeling

3.2.1 Database diagram

The graph below and its legend (see Appendix B.2) present the architecture of our database (https://app.quickdatabasediagrams.com/#/d/tJVTi0): our different tables with the links between them and the fields of each of them.

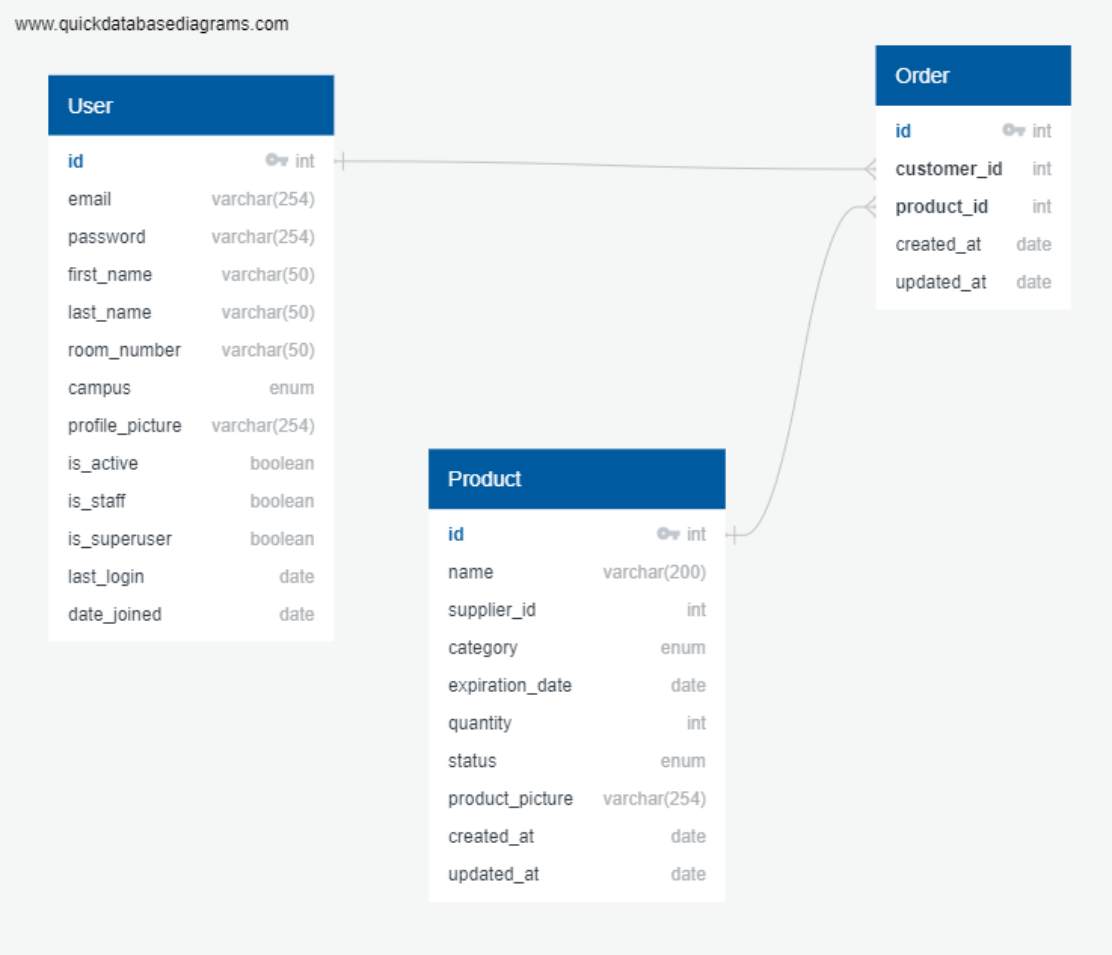


Figure 1: Database diagram

3.2.2 Use case diagram

Below is the use case diagram that provides a black box view of how users/admins can interact with our system. For the sake of readability, extra use cases like *Log Out* and *Change Password* are not represented on this diagram.

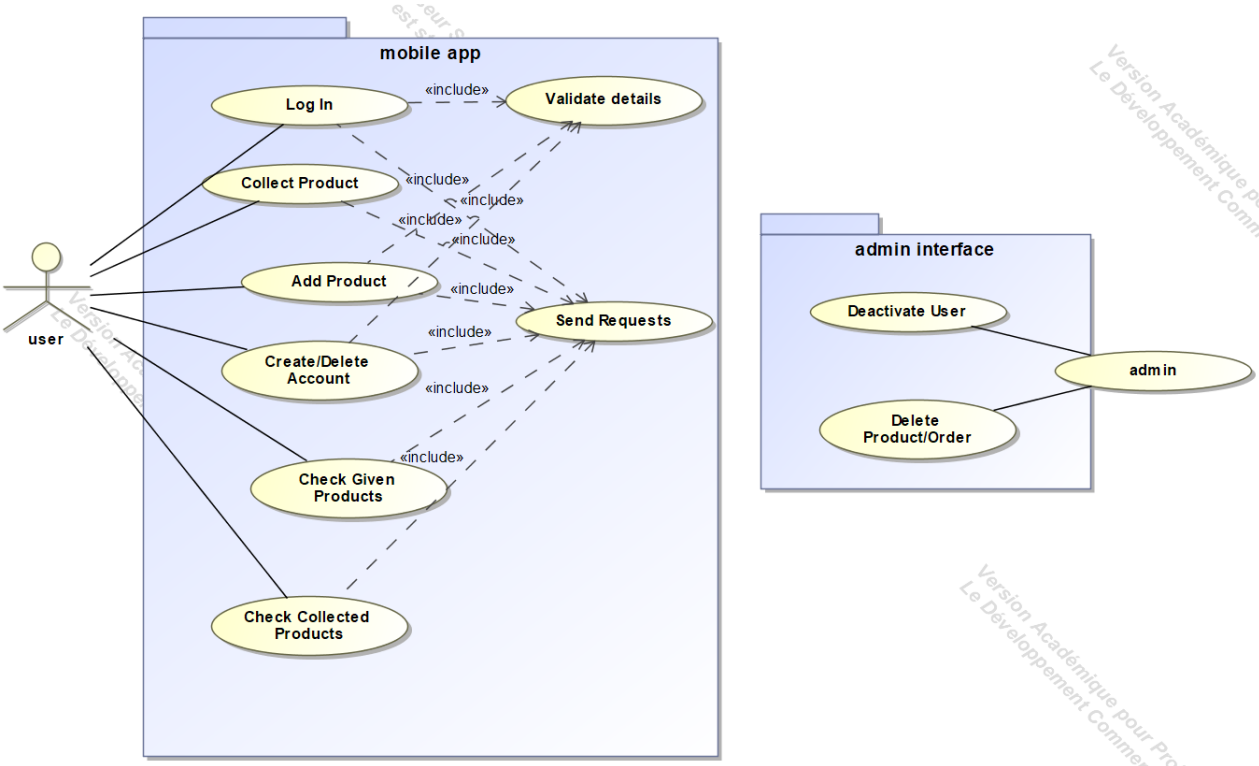


Figure 2: Use Case Diagram

3.2.3 Sequence diagram

To model the behaviour of our mobile application, we made a sequence diagram that will shows object interactions arranged in time sequence. It depicts the objects and the events scenario by showing actions and messages with horizontal arrows. We created several sequence actions to zoom on main functionalities.
Below is the collect sequence diagram that represents the action of collect an available product via the app. For other sequence diagrams, see Appendix C.1

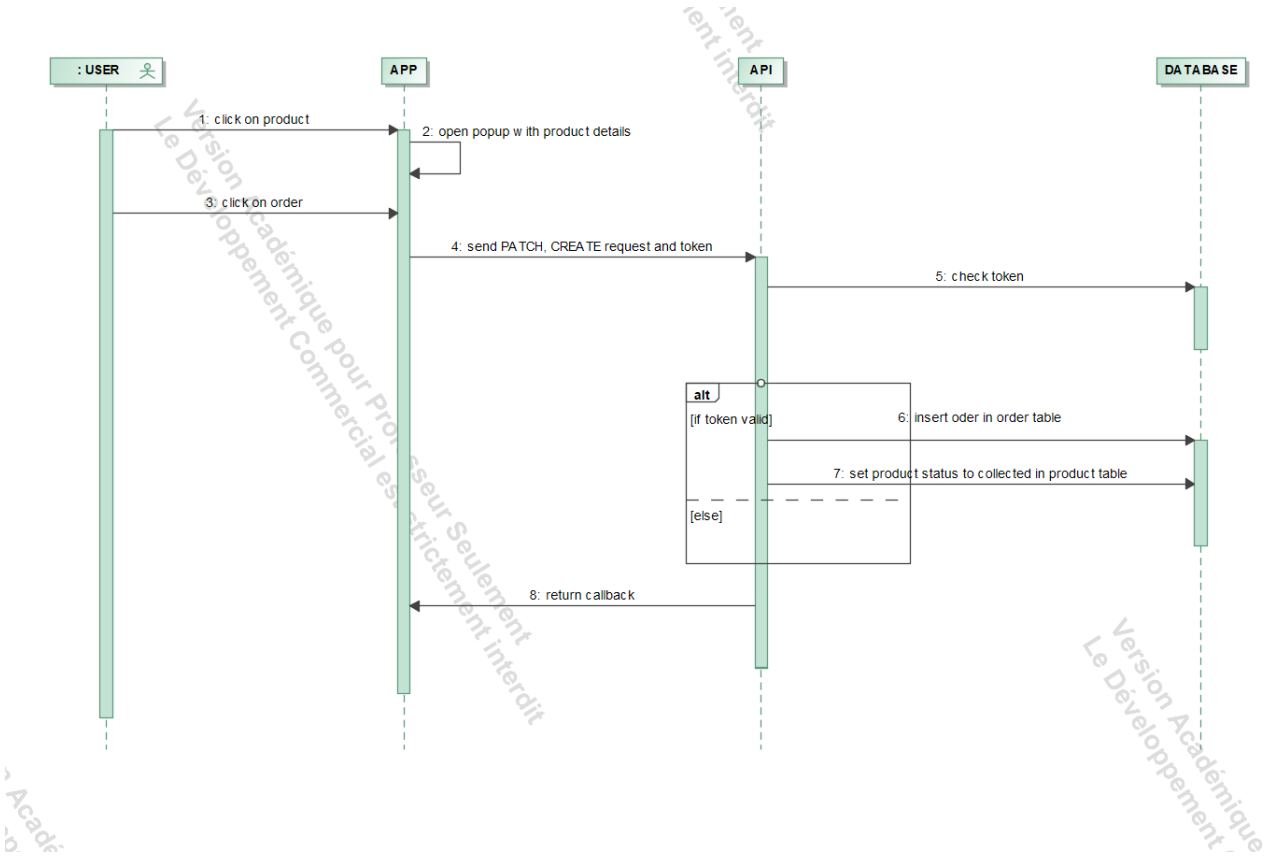


Figure 3: Collect sequence diagram

4 Progress work plan

4.1 MVP

We organized the first half of the project in 4 major steps:
First, we took care to define the scope of the subject (identification of needs, choice of technologies ...).
Then we did a design work in order to define the necessary functionalities. To do this, we first thought about the structure of our database (see Appendix B.2). Then, in order to confirm the satisfaction of the user requirements, we made a functional model to design the user interface. (see Appendix D.1) This allowed us to simulate the a priori functioning of the app according to the initial objectives and to have a first estimation of the development time needed. The development phase first started with the installation and configuration of the server and then with the quick and suboptimal realization of a first version of the application

already equipped with the functionalities of control/update of a product and authentication of a user (see Appendix **??**).

The aim of this first version is not to obtain a finished product but rather to develop a prototype as quickly as possible, on the one hand to become familiar with the techniques to be used and on the other hand to be able to submit the prototype to a panel of users. Once the feedback from the users has been analysed, this time we plan to develop the final version of our product.

4.2 Successes

From our MVP, since we have already had to implement almost all the functionalities proposed by CShare, we were confronted with problems that we were able to overcome by using development tools.

REST API The communication between server and client is done by means of an API (Application Programming Interface) because in the interest of clarity and organization, we wanted to create a well-made method to access particular elements. For this purpose, we chose to use Django REST framework (https://www.django-rest-framework.org/) which is a documented, accessible and powerful tool to build Web REST APIs. The REST framework will allow us to build an API describing the resources used in our Android application. The great advantage of using a REST API is that we have a separation between client and server.

Security ans robustness On the client side, our application is very robust because it has very few security holes as it consists almost entirely of requests to the server. Since we work locally, we have not yet configured our server to support the HTTPS protocol. The only data stored in the Shared Preferences are the token, the user id and the is_logged_in boolean. To prevent the user from accessing those, we should encrypt this key-value data using the Security library, part of Android Jetpack (https://developer.android.com/topic/security/data), that provides an implementation of the security best practices related to reading and writing data at rest, as well as key creation and verification. The Security library contains the following class to provide more secure data at rest: EncryptedSharedPreferences that wraps the SharedPreferences class and automatically encrypts keys and values using a two-scheme method. Keys are encrypted using a deterministic encryption algorithm such that the key can be encrypted and properly looked up. Values are encrypted using AES-256 GCM and are non-deterministic.

On the server side, Django provides a full description of possible security holes. However, we believe that in the time given to us and considering that our application does not contain sensitive data, we have chosen to limit ourselves to deploying the site in HTTPS because without this, it is possible that malicious network users may intercept authentication data or any other information transferred between the client and the server and, in some cases, for active network attackers, that data may be modified in either direction. In addition, as mentioned above, we have developed an authentication method based on Django's Token-based authentication. The token authentication works by exchanging email and password for a token that will be used in all subsequent requests so to identify the user on the server side. Token authentication is suitable for client-server applications, where the token is safely stored. It is important to note that the default Token implementation has some limitations such as only one token per user, no built-in way to set an expiry date to the token but this is still quite acceptable in the case of CShare. We decided to use the library django-rest-auth (https://django-rest-auth.readthedocs.io/en/latest/) that says : *"Since the introduction of django-rest-framework, Django apps have been able to serve up app-level REST API endpoints. As a result, we saw a lot of instances where developers implemented their own REST registration API endpoints here and there, snippets, and so on. We aim to solve this demand by providing django-rest-auth, a set of REST API endpoints to handle User Registration and Authentication tasks."*

HTTP requests Since the major functionalities of the application are based on requests to the server, we had to implement tools to control and detect the content sent by the client during a request and compare it to the content accepted by the server. To do so, we have created a Postman environment (see Appendix D.2) listing all our API endpoints. This way, regardless of the client, we were able to ensure that the requests were valid on the server side. Android side, we quickly realized developing our own type-safe HTTP library to interface with a REST API could be a real pain: we would have to handle many aspects, such as making connections, caching, retrying failed requests, threading, response parsing, error handling, and more. Retrofit, (https://github.com/square/retrofit) on the other hand, is a well-planned, documented and tested library that save us a lot of time. It is dead-simple to use (https://vickychijwani.me/retrofit-vs-volley/). It essentially lets you treat API calls as simple Java method calls, so you only define which URLs to hit and the types of the request/response parameters as Java classes. The entire network call + JSON/XML parsing is completely handled by it (with help from Gson for JSON parsing), along with support for arbitrary formats with pluggable serialization/deserialization. Documentation is great and the community is huge.

Retrofit is also compatible with the HttpLoggingInterceptor class (https://github.com/square/okhttp/tree/master/okhttp-logging-interceptor) of the OkHttp library which logs request and response information whenever the client issues a request.

Image handling Retrofit does not come with support for loading images from the network by itself. So we use another small library called Picasso (https://square.github.io/picasso/). Not only Picasso is just as easy to use as Retrofit, but it also avoids over-burdening the core library.
Moreover, Retrofit has full support for POST requests and multipart file uploads. Indeed, with application/x-www-form-urlencoded, the body of the HTTP message sent to the server is essentially one giant query string — name/value pairs are separated by the ampersand (), and names are separated from values by the equals symbol (=). But for each non-alphanumeric byte that exists in one of our values, it's going to take three bytes to represent it. This is very inefficient for sending large files hence this method cannot be used for sending image files. That's where multipart/form-data comes in. Multipart sends a single object(file) in various parts, each part is separated by a boundary and has some portion of object's data. Each part also has its own headers like Content-Type, Content-Deposition.

Forms and validation Regarding form validators, Django Rest Framework already takes care of setting up some validators on fields but we made the choice to set up validatiors directly from the client side to send the HTTP request only if the form is valid: this avoids useless and time-consuming requests. For this, we use the Android Saripaar library (https://github.com/ragunathjawahar/android-saripaar) which is a simple, feature-rich and powerful rule-based UI form validation library for Android. It is the simplest UI validation library available for Android.

4.3 Failures

OS choice The first thing we regret during the design phase of the project is to have made the choice to develop a native Android application. Indeed, at that time we were only considering deploying the app on the Metz campus where Android is mostly present and in parallel we were taking a course on development under Android Studio. So we were influenced in our choice. However, with hindsight and considering not only Metz but also the Rennes and Gif campuses, where the proportion of people using iOS is not negligible, we might have had to go for a hybrid app to deploy it on both iOS and Android. Faced with this difficulty, we decided to continue on our way and finish the native Android development even if

it means extending the project next year.

App architecture The second big difficulty we encountered appeared when we wanted to specify the architecture of our application by making a class diagram. We immediately noticed that our code was not structured: there were almost no classes apart from activities and models. As a result, a lot of code was duplicated and we did not respect Android's recommendations in terms of app architecture (https://developer.android.com/jetpack/docs/guide#common-principles). So next phase will be trying to rebuild the architecture of our application from scratch while reusing as much of the work done before as possible, relying on reliable sources (see Appendix C.2).

4.4 Development prospects

At this stage of our project, at least we know how to implement the main functionalities. So new challenges are upcoming such as enhancing the user experience. Below are some essential features we want to add to our MVP.

User experience It is important to make an application that is responsive, ergonomic and with a design that fits its context. Apart from the design we plan to focus on later on, we want to provide more functionalities to the users.
An essential one is the search among the available products. A search bar will therefore be implemented in the collect activity to allow the user to search for specific products by entering text.
In addition to this search bar, we also wish to make it possible for the user to filter available products according to their category.
We do not think we'll have time to implement this optional function but it would be great to be able to add a product automatically by directly scanning a barcode. The application would therefore open the camera and scan the code. This would make it possible to add products much faster.

Product reporting For the comfort of users, any non-compliant or irrelevant product should be deleted by the administrator and the malicious supplier should be banned from the platform. To do so, each user and product will be affected with a trust score. The more reports a product will get, the lesser score it will have. Regarding to a predefined threshold, the admin then will have the possibility to remove the user and all his products from the application.

Put users in contact when an order is placed The GDPR (General Data Protection Regulation) https://gdpr-info.eu/ determines what can and cannot

be done when it comes to collect personal data from users. We have several choices for the implementation of the connection between users:

- Connect via an third-party app (message/e-mail),
- Communicate the room of the person offering the product
- Implementing an internal chat in the app.

We thought that the GDPR was going to eliminate some options, but it did not. Indeed, the GDPR leaves us quite free as long as we are transparent with the users. For example, we need to have their consent to use their personal data, to let them know where their data is kept and for what purposes. We must also provide them with the possibility to retrieve their data or delete them from our database. We will therefore make users accept our terms of use and provide an information page about their data in the app. Finally, we retain that we are free to choose between our three tracks for linking. However, the second one does not seem suitable and this will be confirmed by collecting the users' opinion. We will make the final choice by comparing the time we have left and the workload required.

Security Another fundamental step for security that has already been mentioned in this report, is to switch requests from HTTP to HTTPS. This is currently not feasible because the Django runserver command does not support local HTTPS requests. Before any deployment, we'll have to setup the HTTPS protocol.

5 Conclusion

After becoming familiar with all the development tools and after learning and implementing features such as queries, we realized mobile application development requires a lot of organization. It is necessary to have a clear model of the application and to do many iterations to be sure to meet the client needs. On the other hand, the developer has to be meticulous about good practices so that he does not end up with an application that needs to be rebuilt from scratch or that is poorly optimized and then provides a bad UI experience. We also note that we should not hesitate to go in depth through docs during the learning phase because many technologies lighten the development process if they are known.
So far we are proud of our achievements in terms of security and validation of user actions on the client side and the way we manage queries. Nevertheless, hybrid development and optimization of the application architecture remain challenges to be overcome.

A Market study

See the market_study/survey.pdf and the market_study/survey_results.pdf documents.

B Software development

B.1 State of the art

See the state_of_the_art.pdf document.

B.2 Database

See the database_modeling/diagram_legend.pdf and the database_modeling/sql_diagram.png documents.

C Application architecture

C.1 Architecture modeling

See the architecture/sequence_diagrams/*.png files.

C.2 Best practices under Android: MVVM

See the architecture/best_practices/synthesis.pdf document.

D MVP

D.1 Wireframes

See the wireframes/wireframes_v2.pdf document.

D.2 API endpoints

Ask Clara or Babacar to get the Postman link.

D.3 MVP

See the demo/*.webm files.