FYZZ
BACHELOR THESIS

An ecosystem (database, server and app) to subscribe to a free drink and discounts per day

Software Engineering Group
University of Fribourg (Switzerland)

for the Bachelor's degree
by

Ryan SIOW

supervised by:

Prof. Dr. Jacques PASQUIER-ROCHA
Assistant Pascal GREMAUD

Fribourg, UNIFR, 2019

The secret of change
is to focus all your energy,
not on fighting the old,
but on building the new.
— Socrate

Acknowledgements

I would like to thank my family and friends for their support, as well as the software engineering group team, mainly Pascal Gremaud for his availability and help throughout this project. I would also like to thank professor Jacques Pasquier for giving me the opportunity to work on a real application.

Fribourg, 31 July 2019 R. S.

Abstract

This bachelor's work aims to develop an application using a modern technology stack. The business model is based on subscriptions, a model that is becoming more and more widespread.

Part of the project is mainly focused on the realization of the ecosystem, which is why after a chapter on the design of the application from the user's point of view, different technological aspects such as the choice of frameworks and security are studied.

Key words: application, business model, design, ecosystem, framework, Ionic, Cordova, Node.js, npm, Javascript, Typescript, Firebase, Firestore, database, NoSQL, Stripe, Authentication, schedule cloud functions.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Nowadays, more and more companies are opting for a business model with a subscription. In this model, the customer pays a recurring price at regular intervals in order to have access to the company's products or services. This system has several advantages from the point of view of both the seller and the customer.

As for companies, they benefit from this because they are assured of a predictable and constant source of income from registrants for the duration of the subscription. Not only does this greatly reduce business uncertainty and risk, but it also allows for advance payments, while allowing customers to focus on using the service and, therefore, extend the service by signing a contract for the next period, at the end of the current contract.
From the marketing analyst's perspective, this has the additional advantage that the supplier knows the number of currently active members since a subscription generally involves a contractual agreement. This so-called "contractual" system makes customer relationship management much easier because the analyst knows who is an active customer and who has recently withdrawn.

On the consumer side, they will perceive subscriptions convenient if they think they will buy a product on a regular basis and that they could save money. In addition, a subscription model can be beneficial to the customer if it forces the supplier to improve its product. As a result, a psychological phenomenon can occur when a customer renews a subscription, which may not occur in a one time transaction: if the buyer is not satisfied with the service, he or she can simply leave the subscription and find another seller.
This contrasts with many single-use transactions, where customers are forced to make significant commitments due to high prices. Some argue that, the "one-time purchase" model does not encourage sellers to maintain relationships with their customers (after all, why should they care after receiving their money?).
The subscription model should align the customer and the supplier towards common objectives, as both are likely to benefit if the customer receives an added value from the subscription. The customer who receives value is more likely to renew the subscription and possibly at a higher rate.

It is in this context that the idea of an application offering a free drink as well as commercial advantages per day came up. This application, called FYZZ, is the result of careful work on this subscription model that many other companies have chosen, such as Netflix[1] a subscription-based streaming service which offers online streaming of a library of films and television programs, or Spotify[2], an audio streaming platform.
A first part of this report will be devoted to the presentation of the ecosystem from a user's point of view. It is a question of showing here, how the application works as well as showing the whole process: from downloading the application to consuming the service.
In the second part, the realization of the ecosystem will be explained. The architecture, database, server and security will be part of the theme of this chapter.

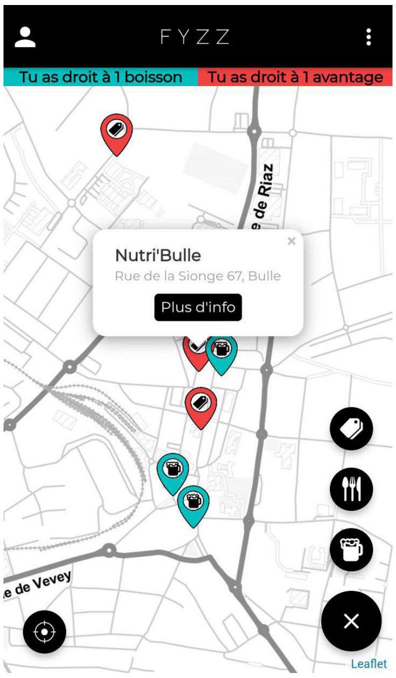
[1]https://www.netflix.com/ch-en/
[2]https://www.spotify.com/ch-fr/about-us/contact/

2 The ecosystem of FYZZ

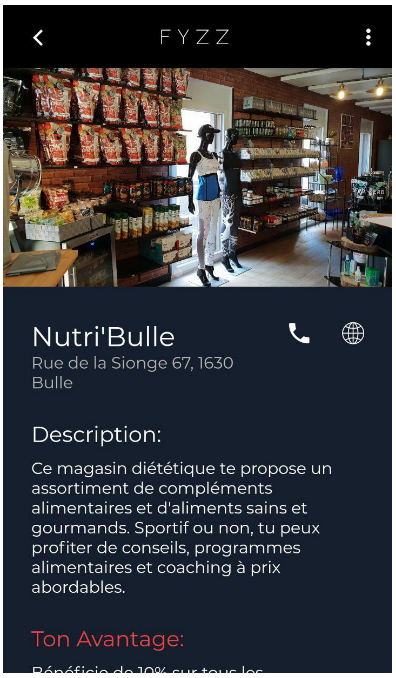
In this chapter we will see the ecosystem from the user's end. Although it is a simple concept, it has to be as straightfoward and userfriendly as possible while reducing to the strict minimum the workload for the business partners. Indeed, the modern society in which we live has very little tolerance for waiting time and complexity. This is why this application has been designed in such a way that the user can enjoy FYZZ services with as few clicks as possible. Figure 2.1 shows the screenshots of the important pages of the application. In fact the only pages needed in order for the customers to benefits from the goods and services.
In order to be able to explain the entire application process, it is advisable to first introduce the actors interacting with each other,and then briefly describe the design.

2.1 Design

Most people using digital media are either on a desktop or a mobile phone. As mentioned earlier, we live in a fast paced society and people are now looking to spend less time to get more done without having any issues to deal with. Therefore, they are rushing into mobile apps because of their functionalities and the ability to get things done quickly. But how can we improve this and make a mobile app better?
The answer might be UI/UX app design. It is nowadays unthinkable to neglect the design of an application as it is an important factor to create engagement. Indeed, all technology companies are constantly innovating the design of their products, both in terms of software and hardware. The design of an app enhances the user experience, which is the biggest factor that ensures the success of an app. To understand the importance of UI/UX app design [12], let us understand the terms well.
What is UI? The abbreviation UI stands for User Interface. It is the way through which users can interact with any mobile apps. User interface design for mobile applications is aimed towards easy, enjoyable and effective interactions between users and the app. The primary goal of UI is to provide the best interaction possible.
What is UX? UX stands for User Experience. The whole idea of UX is to create a system that provides the best experience to the users. The aim of UX mobile app design is to turn cus-



(a) Main page.

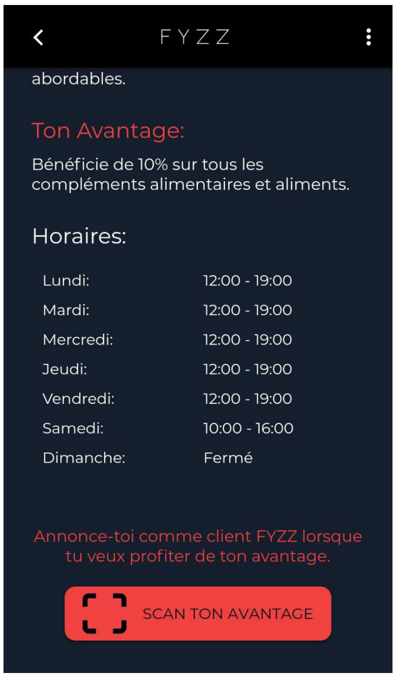

FYZZ

Nutri'Bulle
Rue de la Sionge 67, 1630
Bulle

Description:
Ce magasin diététique te propose un assortiment de compléments alimentaires et d'aliments sains et gourmands. Sportif ou non, tu peux profiter de conseils, programmes alimentaires et coaching à prix abordables.

Ton Avantage:

(b) Shop page top.


FYZZ

abordables.

Ton Avantage:
Bénéficie de 10% sur tous les compléments alimentaires et aliments.

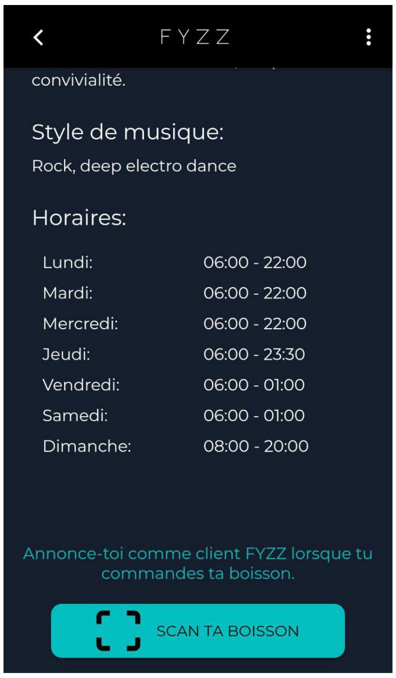
Horaires:

Lundi:	12:00 - 19:00
Mardi:	12:00 - 19:00
Mercredi:	12:00 - 19:00
Jeudi:	12:00 - 19:00
Vendredi:	12:00 - 19:00
Samedi:	10:00 - 16:00
Dimanche:	Fermé

Annonce-toi comme client FYZZ lorsque tu veux profiter de ton avantage.

SCAN TON AVANTAGE

(c) Shop page bottom.


FYZZ

convivialité.

Style de musique:
Rock, deep electro dance

Horaires:

Lundi:	06:00 - 22:00
Mardi:	06:00 - 22:00
Mercredi:	06:00 - 22:00
Jeudi:	06:00 - 23:30
Vendredi:	06:00 - 01:00
Samedi:	06:00 - 01:00
Dimanche:	08:00 - 20:00

Annonce-toi comme client FYZZ lorsque tu commandes ta boisson.

SCAN TA BOISSON

(d) Bar page top.

Figure 2.1 – FYZZ screenshots taken on Android.

tomers into loyal customers by providing a positive experience to them. UX is responsible for a user's journey through the mobile app or website which ultimately decides the success of

the business[6].
Mobile app UI/UX designs are always put together as they are complementary to each other. For an app to be successful the UI/UX app design must be the best.
FYZZ uses iOS UI, respectively android material design to build the user interface on both platforms, iOS and android. The app must evoke the night, the outing, as well as the commercial discounts. That's why the theme is composed of dark colors. The overall design is refined and minimalist with a black and white map as well as a personalized font. In order to facilitate the user experience, FYZZ uses color codes: orange-red for commercial discounts, and blue for the drinks, as Figure 2.1a. On a specific establishment page, only the necessary and relevant information are shown such as the description, the kind of promotion, music, and business hours.

2.2 Interaction between stakeholders

This concept proposes a triangle-shaped human interaction as in Figure 2.2, composed of employees of FYZZ, owners of establishments and clients/users.

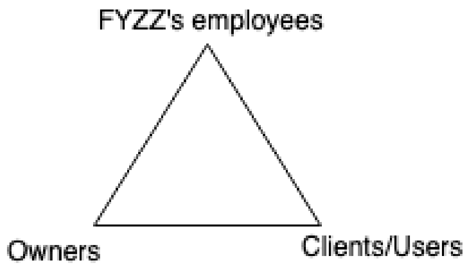


Figure 2.2 – Triangle interaction

Since it implies three actors, three cases of interactions are possible: FYZZ-owners, FYZZ-clients/users, owners-clients/users. These dynamic interactions will be shown in some use-case diagrams in order to fully understand the process of the app. From the clients end, they can perform basic actions on the app such as look at all the Establishments on the map as well as their information, search a specific establishment, pay for a subscription, scan a free drink or a promotion and obtain help within the app. Figure 2.3 shows the full synergy the user has with the app and the other actors. The dotted lines indicates that the communication is implicit, such as paying for a subscription, as this process is fully automated. Indeed, automation is key, therefore physical interaction is needed only when necessary, like the scanning of a QR code.

From the owners end, they view this app as a marketing tool. They can get referenced and listed, which give them more visibility. Owners can also publish notifications, thus attracting potential customers. Once a customer wants to activate their coupon of the day, the establishment has to present the QR code which then need to be verified by an employee through a unique verification code. Bars and restaurants get a refund at a purchase price for every beer,

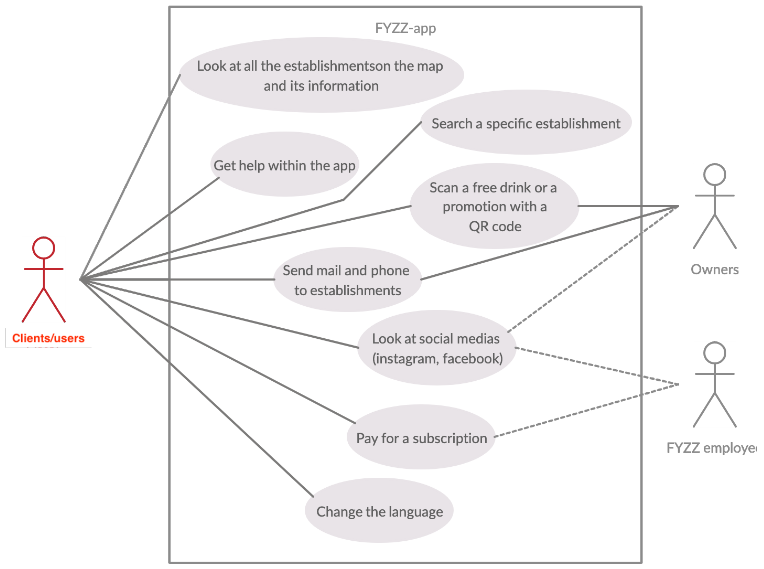

FYZZ-app
Look at all the establishmentson the map and its information
Search a specific establishment
Get help within the app
Scan a free drink or a promotion with a QR code
Send mail and phone to establishments
Look at social medias (instagram, facebook)
Pay for a subscription
Change the language
Clients/users
Owners
FYZZ employe

Figure 2.3 – Use case from clients end: the dotted lines indicate an automated interaction between the stakeholders.

coffee or mineral given through the app. Figure 2.4 shows the communications between the owners and the other actors. Note that all interactions with clients as well as FYZZ employees uses straight lines as nothing is automated yet in this situation. The refund of the beverage is done outside of the app.

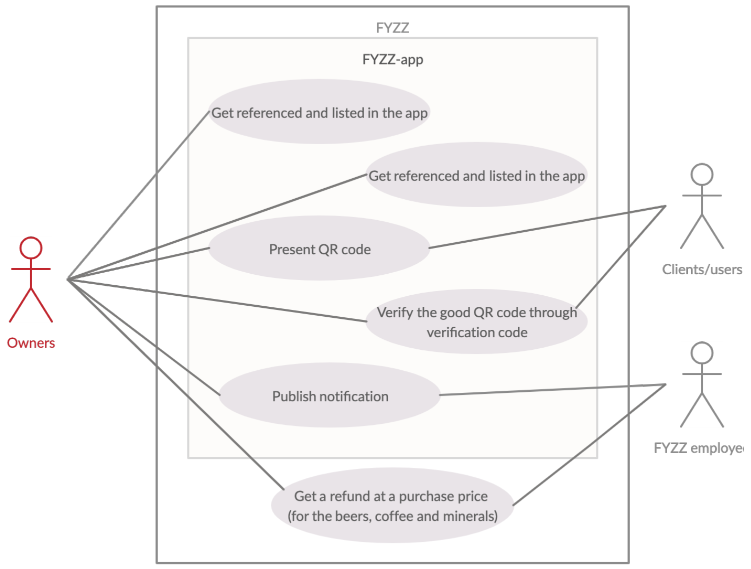

FYZZ
FYZZ-app
Get referenced and listed in the app
Get referenced and listed in the app
Present QR code
Verify the good QR code through verification code
Publish notification
Get a refund at a purchase price (for the beers, coffee and minerals)
Owners
Clients/users
FYZZ employe

Figure 2.4 – Use case 2 from owners end.

2.3 Business Process

Nowadays, everything goes fast and mobile users tend to have less tolerance against complex apps. Therefore, FYZZ is built in such a way that it is easy to use and intuitive. At the very beginning, which is the free downloading of the app through the Apple store and the Google Play store, FYZZ can be viewed in two distinct ways. First, it can be used as a free referencing tool like the well known "local.ch"[1], a Swiss website. Indeed, FYZZ references the establishments in the app which the users can search and filter by category such as bars, restaurants and boutiques. It is only when the customers decide to sign up for a subscription that they can take advantage of the promotions and the free drink offered by the business partners. Consequently, the need of a good transaction process is essential and far too important as it represent the core process of FYZZ and contributes to the benefits of the company.
The transaction process in question is to convert the daily offer the users have to a consumable good through a scan method. This process must be simple and intuitive. First, the client open the app once he or she steps in a referenced establishment. At the very beginning, the app checks the user's personal data stored on a cloud to see whether the offers (one drink and one promotion) are activated for the day. Figure 2.1a shows two banners in blue and red-orange at the top of the map, indicating that offers are available. The user can then navigate to the corresponding establishment page where the scan button is enabled (if the offers are not available, the scan button is then disabled). The user tells to an employee of the establishment that he is a FYZZ customer. Following this step, the worker presents a QR code specific to the merchant that the user scans. The app then check upon scanning the QR code whether it belongs to this establishment. If not, an error alert is prompted and the scanning is cancelled. If the code indeed belongs to this establishment, a verification code alert is prompted and the shopkeeper has to enter his specific four-digit code to validate the scanning. If the four-digit code is incorrect, the manipulation is taken back to the establishment page. If the four-digit code is correct, a greeting alert shows up and the user can now enjoy his promotion or his free drink of the day. While the greeting alert is on screen, the boolean variable of the offer as well as the scanning boolean variable are set to false in the cloud. The scanning button is then disabled on the app (see figure A.4) and the offer of the day will be unavailable until the next day (at 00:01 UTC). Security of these variables are essential as it can not be possible for a user to manually set back the time on his phone in order to scan for a second time in the day. The resetting of those variables will be explained in the next chapter.
It should be noted that the shopkeeper is implied in only one step because it is important to keep the workload of the FYZZ business partners to a minimum. Therefore, the scanning process can be summarized in three major steps: going on the page of the corresponding establishment, scanning the QR code and finally verifying the QR code. The user clicks only two times during the whole process and the shopkeeper enters the verification code. Although it is already a simple process, it can be improved in the future where the QR code could be given to the users (in the app) instead to the establishment. A scanning device could then be needed and would be provided to the business partners in order for this new process to work.

[1] https://www.local.ch/en

In this situation, only a single click is enough, eliminating the shopkeeper's involvement. Due to a lack of time and money, this alternative has been abandoned but remains nevertheless a significant option for a possible future project.
Figure 2.5 shows the full scanning process. The first two pools (clients and establishment) happen in the app whereas the third pool happens in the cloud-server and database.

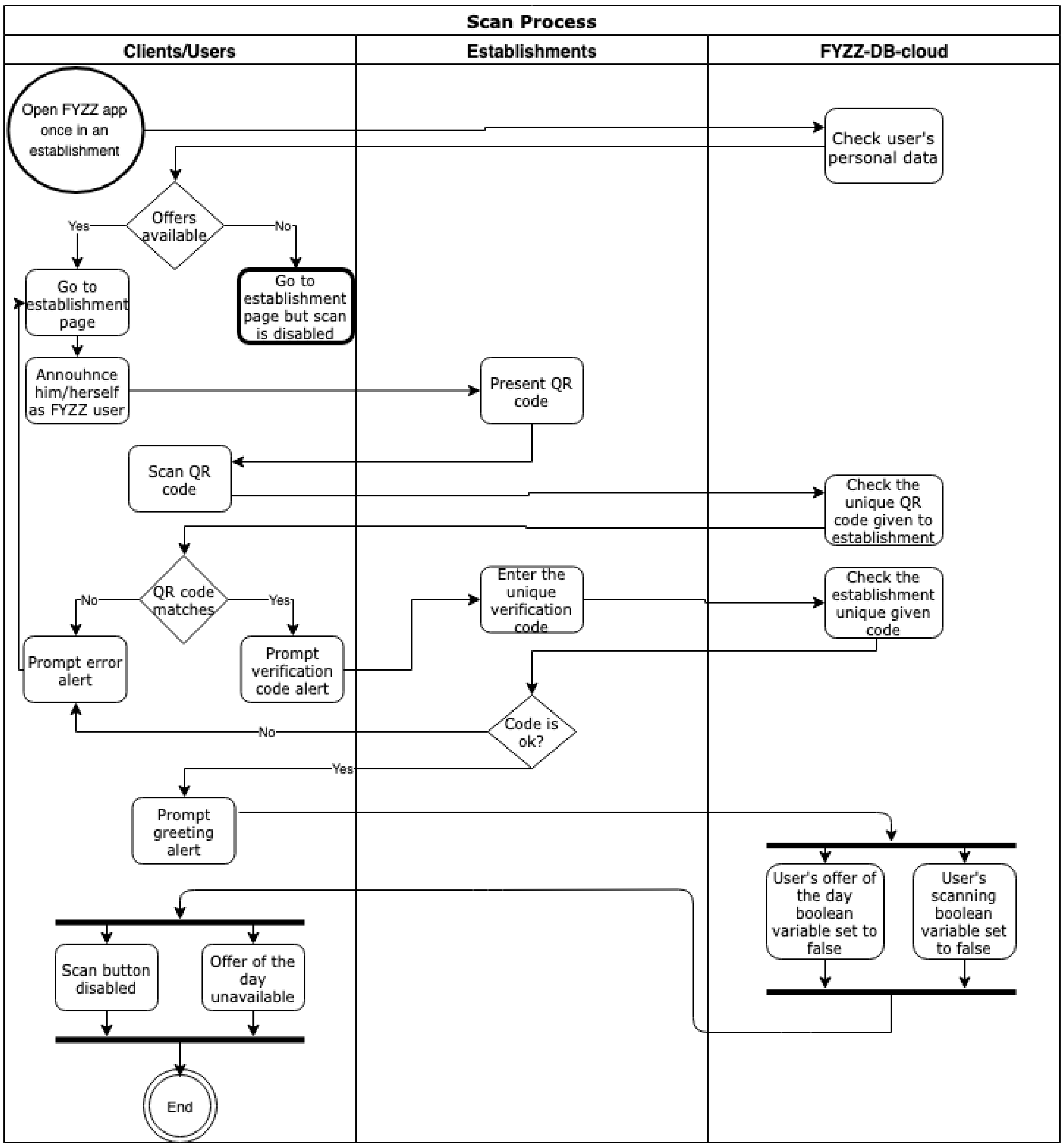

Scan Process
Clients/Users
Establishments
FYZZ-DB-cloud
Open FYZZ app once in an establishment
Check user's personal data
Offers available
Yes
No
Go to establishment page
Go to establishment page but scan is disabled
Annouhnce him/herself as FYZZ user
Present QR code
Scan QR code
Check the unique QR code given to establishment
QR code matches
No
Yes
Enter the unique verification code
Check the establishment unique given code
Prompt error alert
Prompt verification code alert
Code is ok?
No
Yes
Prompt greeting alert
User's offer of the day boolean variable set to false
User's scanning boolean variable set to false
Scan button disabled
Offer of the day unavailable
End

Figure 2.5 – Activity Diagram of the scan process

3 The realization of the ecosystem

This more technical chapter is dedicated to the development of the ecosystem. This is not just a simple app as FYZZ uses a stack of modern technologies. As there are several software solutions and plugins on the market, the design, implementation and operation of such a service require strong cooperation between these different types of integration. Questions like what technology is used, how is the architecture of the database build, how is the scanning process coded and how to prevent time cheat, will be discussed.

3.1 Goal of the project

Let us briefly recall the purpose of this project: Fyzz is a subscription that allows users to enjoy a free drink (coffee, mineral or draught beer) per day. In addition to this privilege, customers can enjoy commercial benefits. The users can look for a specific establishment which is referenced in the application. To validate their offers, users must go in the page of the establishment and validate their offer by scanning a QR code.
In order to target the maximum number of users, it is imperative to be able to build the application on the two largest and well known OS system which are Android and iOS. A database is required to handle the users data as well as the establishments one. Finally, a strong security system is needed to handle the transactions without the user being able to cheat.

3.2 Choice of technology

This concept, although simple, must nevertheless integrate several different framework to carry out this project. There exist multiple solutions to achieve this application, and the choice of frameworks depends only on the programmer's affinity with them. Figure 3.1 shows all the technologies used and is divided into four parts: the text editor and all the programming languages used, the mobile application development frameworks, the front-end mobile optimized library, and the database.

The preferred text editor is atom (see Appendix A.1) since it supports plugins written in Node.js and was developed by Github which is good for versioning.

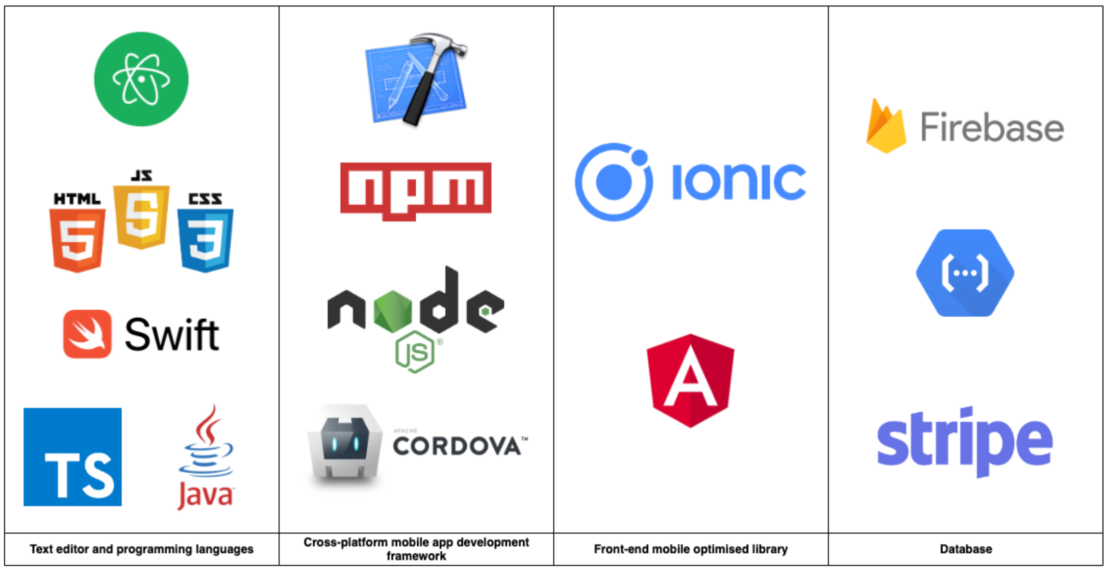

HTML
JS
CSS
Swift
TS
Java
npm
node JS
CORDOVA
ionic
Firebase
stripe
Text editor and programming languages
Cross-platform mobile app development framework
Front-end mobile optimised library
Database

Figure 3.1 – Technologies used

3.2.1 Ionic Cordova

Because the application needs to be on both iOS and Android platforms, and to shorten development time, Ionic[4] and Apache Cordova[1] are chosen. Apache Cordova is a mobile development framework that allows developers to create hybrid mobile apps using web technologies like HTML, CSS and JavaScript, meaning that they are neither truly native mobile application (because all layout rendering is done via Web views instead of the platform's native UI framework) nor purely Web-based (because they are not just Web apps, but are packaged as apps for distribution). Hybrid apps are hosted inside native applications that allow them to access to native device APIs such as the device's camera, pedometer and other functionalities, removing the need to develop for a specific operating system separately. Ionic on the other hand is a front-end, mobile-optimized library that can be used to make Cordova applications look native. It is in fact build on top of Cordova so it can use the ability to access the native device's APIs (Appendix A.2 shows all the plugins used in the project). Other alternatives like Flutter (Google's own mobile development framework), React Native or just Native all have their own benefits. But for the sake of time and cost efficiency, Ionic Cordova was chosen instead. Table 3.1 shows the comparison between a software development kit (SDK) such as Ionic and Native coding.
To execute Javascript code outside of a browser, the installation of Node.js[8] is necessary. It is an open-source, cross-platform JavaScript run-time environment which comes with its default package manager npm[9] (originally short for Node Package Manager). Appendix A.3 shows the information about the project setup as well as the system and environment.

Table 3.1 – Ionic vs Native.

	Ionic	**Native**
Tech knowledge required	HTML, CSS, JavaScript/-Typescript	Swift / (iOS), Java (Android)
create versions for other OS?	**Yes** - Ionic can run the same code regardless of the platform.	**No** - need to build a new version from scratch.
Can access native features	Most native features can be accessed with Cordova plugins.	**Yes** and new features are supported from day 1.
Performance	Can feel clunky but this can be mitigated with good design practises.	Will create the smoothest and fastest feeling apps as there are no layers of abstraction in code.
Reliability	Relies on third party plugins which save on development time, but any issues with a plugin will cause issues with the app.	The most reliable, however it will need experienced developers to do maintenance for each operating system.
Time and cost	The quickest and cheapest.	Each version for each operating system will cost a similar amount to create and maintain – no shortcuts.

The main programming languages used are HTML, Javascript, Typescript and CSS for page rendering. Although Cordova uses plugins to access native device functionalities, some of them need specific authorization that can be specified only in the native files. Therefore the use of Swift and Java was necessary.

3.2.2 Firebase and Firestore

Now that the mobile frameworks are chosen, a database is needed to store the client's and establishment's data. This database will also handle the data of the scanning transaction as well as the process of it. Firebase is Google's mobile and web application development platform. It proposes services[1] such as authentication, Firebase Storage, cloud Firestore, cloud functions and many more. Cloud Firestore is discussed in this subsection.
Cloud Firestore[2] is what is known as a NoSQL database like MongoDB[2] which some people are familiar with. In traditional relational databases like MySQL[3] or mobile frameworks like

[1]https://firebase.google.com/products
[2]https://www.mongodb.com/what-is-mongodb
[3]https://en.wikipedia.org/wiki/MySQL

SQLite or Core Data[4], every table has its own schema, which means that every row in that table is strictly defined. A specific set of columns can be added per row, and every column has its own very strict rules about what kind of data type goes in there. Because of this very strict schema, people usually end up storing one type of object per table in their database. And if someone wants to associate one object with another object hanging out in another table, the person is usually doing that by creating another column known as a foreign key that contains the unique ID of that other entry in that other table.
In the NoSQL world, things are different. In general, all the data are not stored in neat tables like in SQL. In fact, there is a number of different ways of how they can be stored: from a key value storage, to a nested tree or to a collection of JSON objects. But one thing that most of them have in common is that NoSQL databases are usually schema-less, which means there is no database-level restrictions around what kind of data one can put at any point in the database. This approach seems odd at first but it does have some advantages. Working with a schema-less structure means that the database design can easily be iterated by adding or changing fields as needed without breaking. It can also come in handy in other situations where other data are stored and similar to another but not exactly the same[7]. For example, FYZZ can easily include tattoo parlors and skydiving lessons in the bars section and the database would not care that one object has a tandem-jumps field and another has a tattoo-style field. The drawback is that the application will need to be coded defensively. While security rules can be set up to help enforce what kind of data are put at what place, there is not a guarantee at the database level that a certain set of data will be retrieved at any time. That means that some checking on the client side are needed to make sure the desired data is really what is expected and fail nicely when it is not. Nevertheless, defensive coding is a good habit to have, particularly in the mobile world, where it is not always guaranteed that users will be running the latest version of the application against the latest iteration of the database. Another significant advantage of a NoSQL database over traditional databases is that it is able to distribute its data across multiple machines fairly easily. Indeed, with most relational databases, if the application gets popular and the database needs to be scaled up to a larger data set over time, it generally need to be put on bigger and more robust machines. This is known as vertical scaling. On the other hand, with many NoSQL database like Cloud Firestore, if the database needs to be scaled up to a larger data set, it can distribute that data across several servers. This is known as horizontal scaling. It is for these reasons, a NoSQL structure was chosen to build the database of FYZZ.
Cloud Firestore meets the criteria mentioned above. It is a collection of documents which are stored in a tree-like hierarchical structure like in Figure 3.2. These documents are similar to JSON objects or dictionaries. They consist of key value pairs, which are referred to as fields in the Cloud Firestore. The value of these fields can be any number of things, from strings, to numbers, to binary data, to smaller JSON looking objects, which the developers of Firebase has named "maps". However, there are a few rules to follow:

- Collections can only contain documents and nothing else

[4]https://en.wikipedia.org/wiki/Core_Data

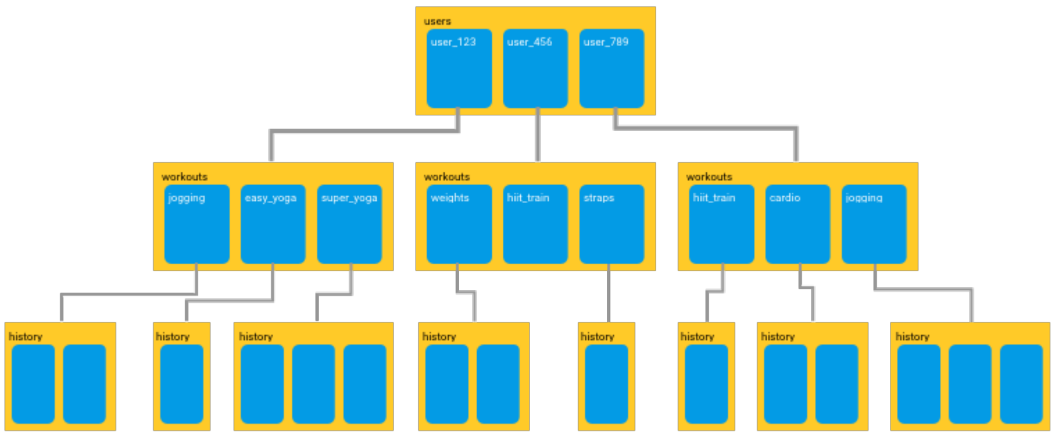


Figure 3.2 – An example of the Firestore structure *(taken from https://proandroiddev.com/working-with-firestore-building-a-simple-database-model-79a5ce2692cb)*

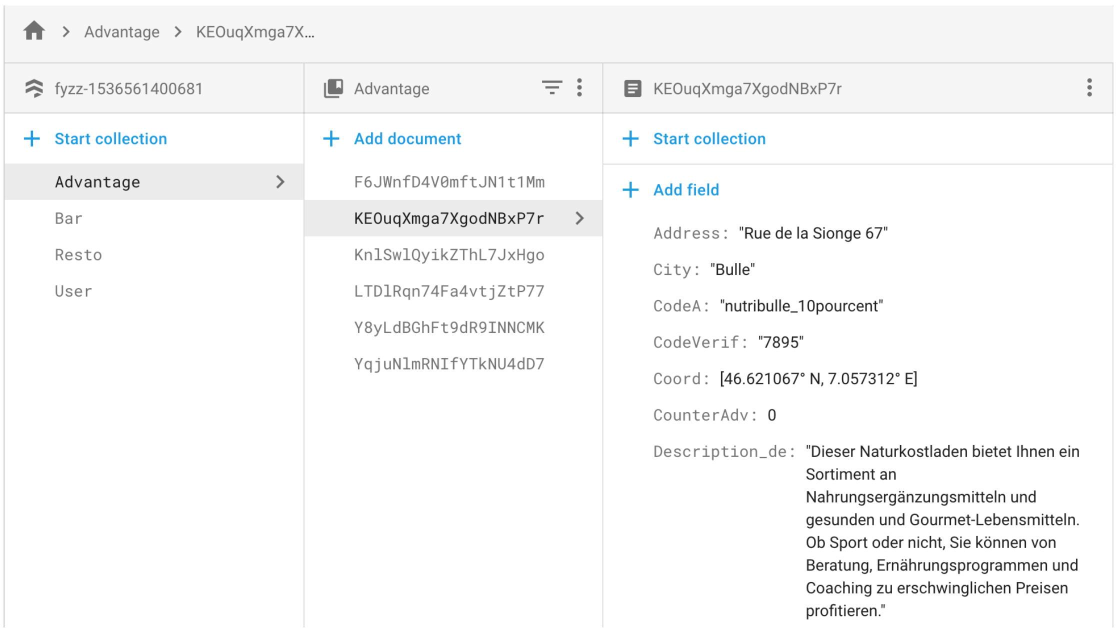
- Documents can not exceed 1Mb in size
- A document can not contain another document. It can however point to sub-collections, but not other documents directly. (The following is possible: a collection containing documents, which then point to sub-collections that contain other documents, so on and so forth.)
- The root of the tree can only contain collections.

As a general rule, one will be drilling down into the structure by specifying a collection, then a document then a collection, then a document, and alternating like to obtain a document containing the desired data. Since this code can be messy, it is better to specify the document or collection by creating a path to that document like so:

Listing 3.1 – Usage of path

```
    firestore.document("User/{userId}/workouts/super_yoga")
```

In the application FYZZ, the database has the following structure. The root is composed of four collections which are "Advantage" (collection of merchants), "Bar", "Resto" and "User". The first three collections share the same configuration which can be seen in figure 3.3a. The "User" collection is a bit different having three other sub-collections ("Source", "Stripe" and "Token") tied to each document like in Figure 3.3b. Note that every document here have a generated unique ID.
Now that FYZZ has a good data structure by adopting a NoSQL type of database, the application

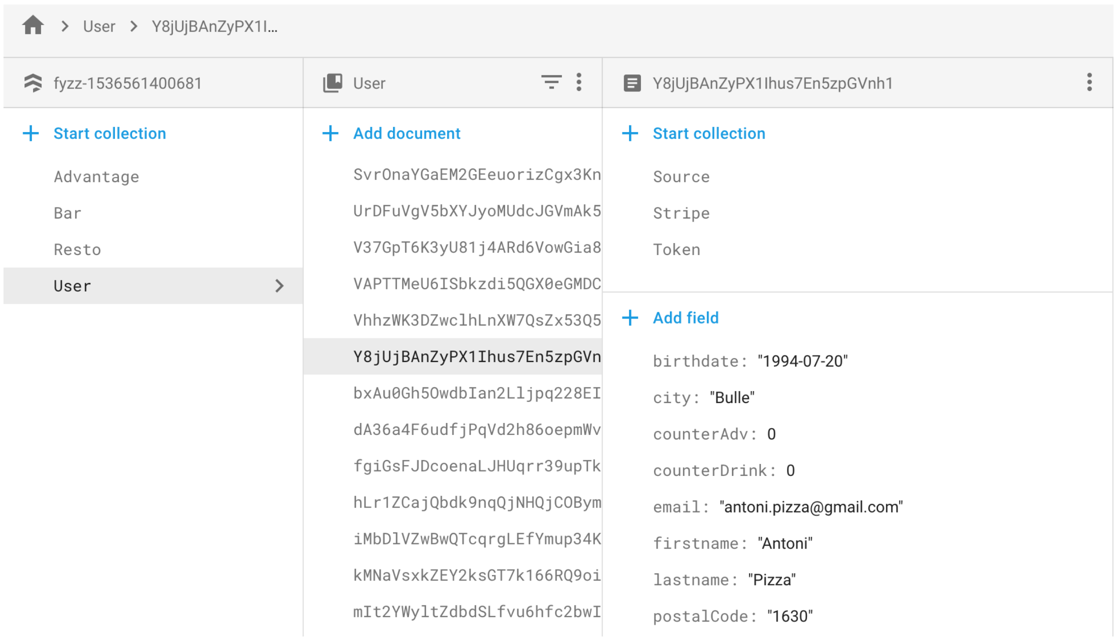


Advantage
Bar
Resto
User

F6JWnfD4V0mftJN1t1Mm
KEOuqXmga7XgodNBxP7r
KnlSwlQyikZThL7JxHgo
LTDlRqn74Fa4vtjZtP77
Y8yLdBGhFt9dR9INNCMK
YqjuNlmRNIfYTkNU4dD7

Address: "Rue de la Sionge 67"
City: "Bulle"
CodeA: "nutribulle_10pourcent"
CodeVerif: "7895"
Coord: [46.621067° N, 7.057312° E]
CounterAdv: 0
Description_de: "Dieser Naturkostladen bietet Ihnen ein Sortiment an Nahrungsergänzungsmitteln und gesunden und Gourmet-Lebensmitteln. Ob Sport oder nicht, Sie können von Beratung, Ernährungsprogrammen und Coaching zu erschwinglichen Preisen profitieren."

(a) Shops (establishment) collection.



Advantage
Bar
Resto
User

SvrOnaYGaEM2GEeuorizCgx3Kn
UrDFuVgV5bXYJyoMUdcJGVmAk5
V37GpT6K3yU81j4ARd6VowGia8
VAPTTMeU6ISbkzdi5QGX0eGMDC
VhhzWK3DZwclhLnXW7QsZx53Q5
Y8jUjBAnZyPX1Ihus7En5zpGVn
bxAu0Gh50wdbIan2Lljpq228EI
dA36a4F6udfjPqVd2h86oepmWv
fgiGsFJDcoenaLJHUqrr39upTk
hLr1ZCajQbdk9nqQjNHQjCOBym
iMbDlVZwBwQTcqrgLEfYmup34K
kMNaVsxkZEY2ksGT7k166RQ9oi
mIt2YWyltZdbdSLfvu6hfc2bwI

Source
Stripe
Token

birthdate: "1994-07-20"
city: "Bulle"
counterAdv: 0
counterDrink: 0
email: "antoni.pizza@gmail.com"
firstname: "Antoni"
lastname: "Pizza"
postalCode: "1630"

(b) Users collection.

Figure 3.3 – Screenshots of the Firestore console.

needs a good way to fetch the data without blocking the code to wait for a result. Luckily, and because the engineers on the Firebase SDK are putting great effort into making their APIs consistent and easy to use, the calls that deal with reading and writing data are fully asynchronous.
In computer science, a synchronous execution means that the code execution will block for

the API call to return before continuing (all the executions are treated by the same thread). The definition can be simplified as a code that is executed in sequence – each statement waits for the previous statement to finish before executing. This means that until a response is returned by the API, the application will not execute any further. An asynchronous execution on the other hand, do not block for the API call to return from the server. All the code is still executed on a single thread for the whole application, but the I/O operation are delegated to other threads while the main thread executes the code of another request. This means that the execution continues on in the program, and when the call returns from the server, a "callback" function is executed [11].
In short, using a synchronous call for this type of project is not only bad but dangerous too. Indeed, stopping the code to wait for an execution that could take a long time must be avoided. Specially in an application runtime environment where the main thread is controlled by an event loop, which goes through a queue of work items forever until the application process dies. These items include handling events, rendering to screen, animations and many other things related to the UI. If one of these work in the event loop takes too long, the application will appear jerky or even completely stuck. Therefore it is critical to make sure the main thread never blocks, so the UI is always smooth and responsive[13]. Using asynchronous calls means that the functions will return immediately without blocking the code to wait for a result. It is then possible to call the functions on the main thread without worrying about the performance and to keep the app responsive, and reducing waiting time for the user. Here is an example of a piece of code extracted from the project:

Listing 3.2 – Asynchronous call with async/await

```
async loadData(){
    let query: any;
    barCode: string;
    ...
    console.log("1. The bar code = " + this.barCode);
    try{
        query = await this.barInfoCollection.ref.where('Name', '==', this
          .barInfo.Name).get().then(querySnapshot =>{
            querySnapshot.forEach(doc => {
                this.barBeer = doc.data().CodeB;
                this.barCoffee = doc.data().CodeC;
                this.barMineral = doc.data().CodeM;
                this.barCode = doc.data().CodeVerif;
                this.barId = doc.id;
                console.log("2. The bar code = " + this.barCode);
            }).catch(error1 => {
                console.log("Error getting document: ", error1);
            });
        }).catch(error2 => {
            console.log("Error getting query: ", error2);
        });
    } catch (err) {
        console.log(err);
    }
    console.log("3. The bar code = " + this.barCode);
    ...
}
```

Three logs (line 5, 14 and 24) are implemented here to show how the asynchronous mechanism works. Let the "CodeVerif" on the database side be "1234". One would expect the log to be:

1. The bar code = unknown
2. The bar code = 1234
3. The bar code = 1234

However, the document fetch method get() in line 4, is asynchronous, and it returns immediately, before the callback that handles the query snapshot is invoked. Then that callback will be invoked later on the main thread, so it can safely update the UI if necessary. This means the third log executes immediately after the first one, before the callback is invoked with the second log like so:

1. The bar code = unknown
3. The bar code = unknown
2. The bar code = 1234

The third log has an unknown value due to the fact that it is executing before the callback where the second log is. Notice the use of async/await[5] syntax. It allows for a clean and concise codebase with fewer lines of code and less typing. In addition, error handling with try/catch is in one place, rather than in every call. Finally, to use this Firebase APi, one must think asynchronously.

3.2.3 Stripe

Because FYZZ is a subscription based consumming service, it needs to have an automated payment process in the application. Ionic Cordova proposes a number of free APIs for payment processing such as Alipay, Apple Pay, Paypal and Stripe. As the goal is to target iOS as well as Android users, Apple Pay can not be a wise choice. The same goes with Paypal since not every body has a Paypal account. To simplify the payment process, only the credit card of the users is needed without the need of creating an account. Stripe comes in handy to this situation. It provides an API that web developers can use to integrate payment processing. To be able to use Stripe, FYZZ must be registered as a company and proves through various documents that it is indeed a real business or business to-be. A secret key is then given.
In the application, when the users want to subscribe, a payment form pops up like in Figure A.5. The users then enter their credit card information which is sent directly to the Stripe API. Upon receiving the data, Stripe sends back a token which is then stored in Firestore in the user's respective document. The credit card information never transit to the FYZZ back-end. The

only information sent to the database is the token retrieved from Stripe. This token, alongside with the secret key, is used to generate the customer_id in the payment server. The token is then sent to the Stripe API in order to create the customer on the Stripe server. This server stores the customer_id associated with the user. This customer_id can later be used to charge the client. Figure 3.4 shows the sequence diagram of the payment process.
The security aspect of the process payment will be discussed in the next section.

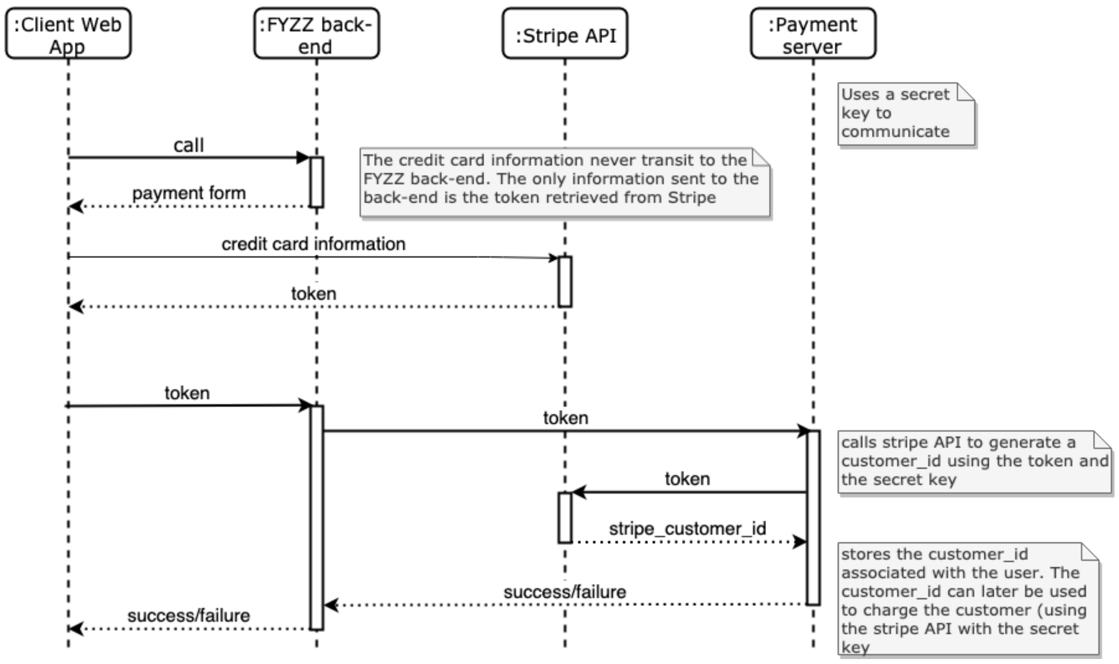

:Client Web App
:FYZZ back-end
:Stripe API
:Payment server
Uses a secret key to communicate
call
payment form
The credit card information never transit to the FYZZ back-end. The only information sent to the back-end is the token retrieved from Stripe
credit card information
token
token
token
calls stripe API to generate a customer_id using the token and the secret key
token
stripe_customer_id
stores the customer_id associated with the user. The customer_id can later be used to charge the customer (using the stripe API with the secret key
success/failure
success/failure

Figure 3.4 – Stripe sequence diagram

3.3 Security

Security is a big concern in this project as it handles users data and transaction that needs to be safe. FYZZ has several data layers to keep secured at all cost: these are the stripe transactions, the Firestore read and write permission, and the scanning process. But let us first of all take an interest in authentication, an aspect not to be neglected.

3.3.1 Authentication

Nowadays, the majority of people carry a mobile phone and there exist a number of way to sign up for services: email, phone number and social account like Google, Facebook or Twitter. The question is what would be a good option to choose for this type of project as there exist multiple solutions. This choice will provide a first security barrier to prevent fake account. To make the choice relevant, let us have a look at the users data that is stored in Firestore: the birth date, the city and its postal code, the first name and last name, the sex and

finally the email address. As FYZZ does not propose a sensitive service involving personal data to be shared, only the authentication through the email is enough. Other than preventing fake accounts, it prevents people from using other people's email addresses and enables a password/login retrieval system. The drawback is that there is no way of knowing if the email address used is just a throwaway address. Using the phone number as an authentication solution is a good alternative but can be intrusive as some people do not want their phone number to be shared. Indeed, an application must have good reasons to ask its users to verify a number, as it can be part of privacy laws. As only the necessary data mentionned aboved are needed, the email verification only is 'justified' in that case.
The following steps show how the email verification works in FYZZ:

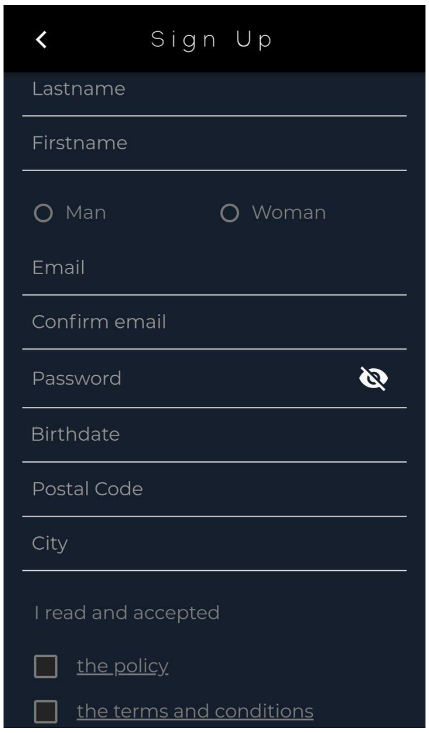
1. The user sign up to create an account (see fig 3.5a).
2. A verification link is sent to the user's email (see fig 3.5b).
3. The user click the verification link and is good to go.

The sign up form (as well as the login form) captures the users credentials which are then sent securely to Google's server. Upon receiving the credentials, Firebase invalidates them, and sends back an authentication token to the users so it is possible to access the data in the front-end from this token such as the name or the email of the user that has just logged in or signed up. When requests are made to the Firebase server after login for example, then this token is sent along. So when a request is made to change some kind of data in the Firestore, the server will be able to look at the authentication token of that request and handle the data based on that user's token[10].

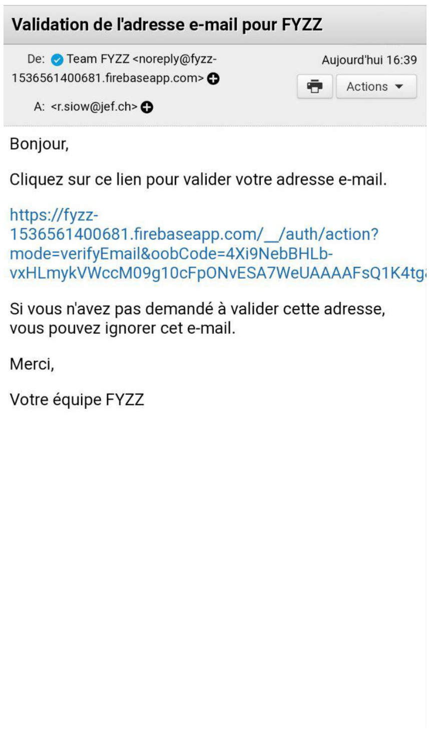
3.3.2 Stripe's security

Another important security point is the subscription itself. Indeed, the users must enter their credit card information, which is a highly sensitive data. To limit the liability that would be related to FYZZ, the credit card information never transit to the FYZZ back-end (see figure 3.4). Therefore, FYZZ relies on the security of Stripe on this aspect. As security is one of their biggest consideration, here is a brief explanation of the security at Stripe[14].
It forces HTTPS (Hyper Text Transfer Protocol Secure a protocol to secure the communication between two system) for its service using TLS (Transport Layer Security, a cryptographic protocol designed to provide communications security over a computer network). The Stripe libraries connect to Stripe servers over TLS and verify TLS certificate on each connection. All card numbers are encrypted with AES-256[5] (The Advanced Encryption Standard with a key length of 256bits). Decryption keys are stored on separate machines. None of Stripe's internal servers are able to obtain plaintext card numbers. Stripe's infrastructure for storing, decrypting, and transmitting card numbers runs in separate hosting infrastructure, and does not share any credentials with Stripe's primary services.

[5]https://en.wikipedia.org/wiki/Advanced_Encryption_Standard



(a) FYZZ sign up form.



Validation de l'adresse e-mail pour FYZZ

De: Team FYZZ <noreply@fyzz-1536561400681.firebaseapp.com>   Aujourd'hui 16:39

A: <r.siow@jef.ch>

Actions

Bonjour,

Cliquez sur ce lien pour valider votre adresse e-mail.

https://fyzz-1536561400681.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=4Xi9NebBHLb-vxHLmykVWccM09g10cFpONvESA7WeUAAAAFsQ1K4tg

Si vous n'avez pas demandé à valider cette adresse, vous pouvez ignorer cet e-mail.

Merci,

Votre équipe FYZZ

(b) Email verification link.

Figure 3.5 – Email authentication.

3.3.3 Database access

All the information of the establishment and users on the application are stored on Firestore. Upon the creation of a FYZZ account, the user create a document on the database. Once signed up, the user can read the information of the establishment fetched on the database. Thus there are two mechanism that exist here which are read and write. Because we are never too careful, it is better to protect the infrastructure with security rules to provide access control. Firebase API proposes a system that consist of match statement which identify documents in the database, and allow expressions, which control access to those documents. In the application, the users can read all the establishments information but they can write/modify only their own data related to their profile. Here are the security rules that match the "User" collection:

Listing 3.3 – Firestore security rules

```
service cloud.firestore {
  match /databases/{database}/documents {

    ...

    match /User/{userId} {
      allow read: if true;
8        allow update, delete: if request.auth.uid == userId;
9        allow create: if request.auth.uid != null;
10
11       match /Source/{source} {
12         allow read, update: if request.auth.uid == userId;
13         allow create: if request.auth.uid != null;
14       }
15       match /Stripe/{source} {
16         allow read, update: if request.auth.uid == userId;
17         allow create: if request.auth.uid != null;
18       }
19       match /Token/{source} {
20         allow read, update: if request.auth.uid == userId;
21         allow create: if request.auth.uid != null;
22       }
23     }
24   }
25 }
```

On line 8, "request.auth.uid == userId" specifies that only this user that is logged in can update and delete the its own document.

3.3.4 Transaction - scanning process

The hardest part of the project is to find a solution to counteract users cheating with the time. Indeed, since FYZZ allows one free drink and one promotion per day only, once the user have activated the offer by scanning the QR code related to it, the scanning button is disable until the next day at 00:01 UTC. How can the program ensure that when the users set the time back in their phone settings, the scanning button would not be enabled again?
It is quite impossible to use a solution without involving a server. Indeed, if all the data and the system is held in the users device, they can easily change the phone settings or access their application data. Here are some solutions to keep the scanning button disabled until the next day, all involving a server:

- **Synchronizing data with the server:** the idea here is to synchronize the data with the server from which the application gets a timestamp and a key. The application should synchronize with the server at least every minute in the case of FYZZ (for example if a user decides to take a drink at 23:59 and the scanning has to be enabled again the next day at 00:01). When synchronizing again, the application sends the current timestamp with the key to the back-end server which then compare the timestamp. With some mathematical functions, the server can reproduce the real timestamp. This solution is extremely greedy as it has to compute every minute. With a database that grows overtime with more users signing up, it can not be an efficient solution.

- **GPS timestamp from geolocation:** a better solution would be to use the timestamp from geolocation since GPS signals has their own time. But again, the server must keep track of the time. Furthermore, there exist several Cordova plugins, where the timestamp are locally generated (from the device). Only one plugin worked where the timestamp comes from the location but the problem was that it did not work with the latest iOS and Android sdk. This solution works well in cities because the phone network is strong, but it can be lagging in the countryside. As the previous solution, it can be expensive in terms of computation cost.

- **Trusted timestamp[16]:** a complicated but efficient solution. The idea it to prove the existence of certain data before a certain point without the possibility that the owner can backdate the timestamp. Indeed, upon querying the server, the timestamp is signed with a hash function so a counterfeit would be easy to detect. Figure 3.6 shows how to get a timestamp from a trusted third party. In addition to being difficult to code, a trusted third party time stamping authority which issues the timestamp are not free.

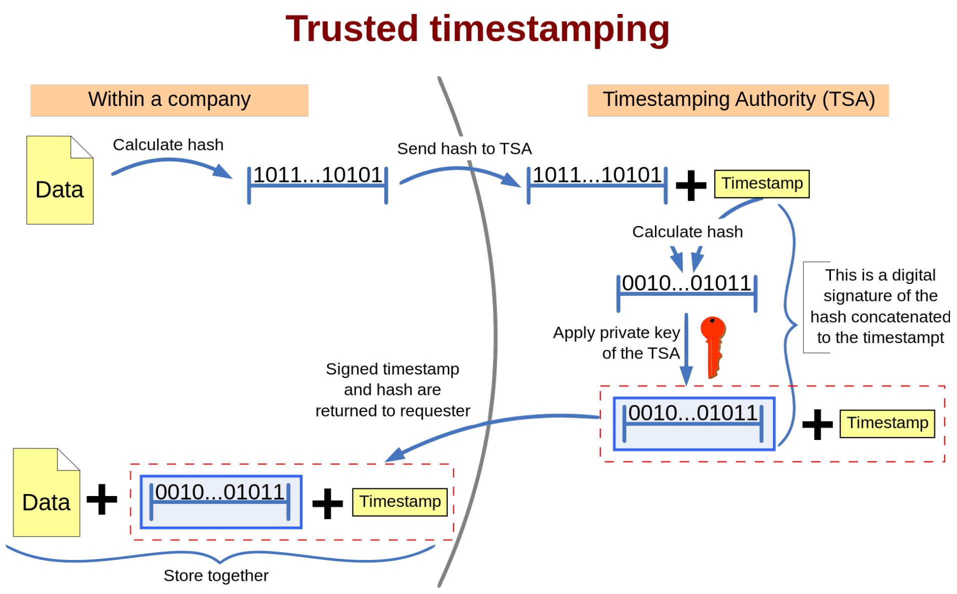
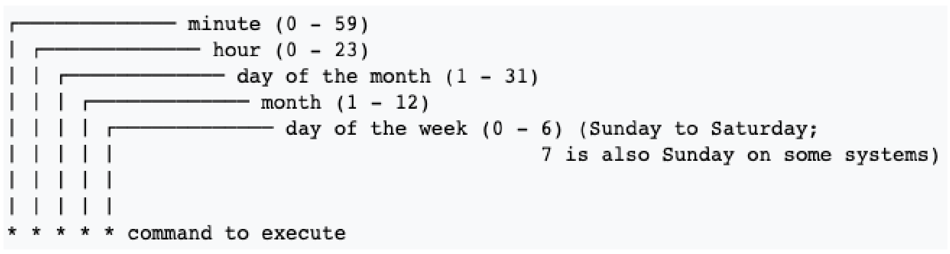


Figure 3.6 – Trusted timestamp (taken from https://en.wikipedia.org/wiki/Trusted_timestamping

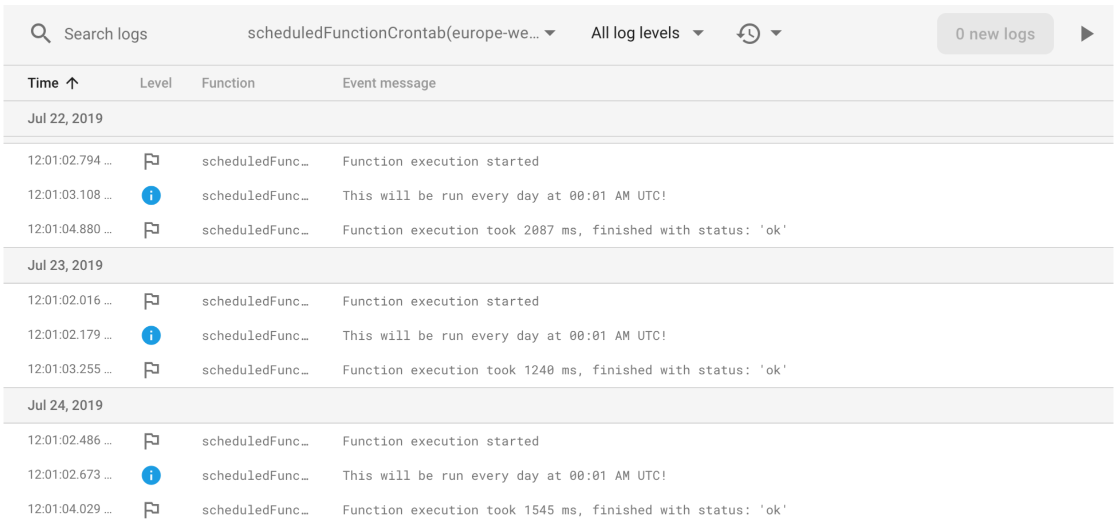
- **Schedule cloud functions (cron job):** a cron[15] is a time-based job scheduler. A cron job is set to run periodically at fixed times, dates or intervals. Since FYZZ uses Google API, there exist a solution to schedule functions using Firebase tools. Similar to a cron job, Google Cloud Functions[3] allows to schedule HTTP requests to functions that is deployed on the server. Indeed, after the code is written and deployed on Google's servers, it is possible to trigger the function directly with a HTTP request set by a scheduler using a cron syntax (see figure 3.7). This solution is free, efficient and do not requires complex coding skills. Among the proposed solutions, this is the best in terms of time efficiency and still delivers a good security as one has to hack Google's server in order to get the data, which is not easy.



```
┌───────────── minute (0 - 59)
│ ┌───────────── hour (0 - 23)
│ │ ┌───────────── day of the month (1 - 31)
│ │ │ ┌───────────── month (1 - 12)
│ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;
│ │ │ │ │                                   7 is also Sunday on some systems)
│ │ │ │ │
│ │ │ │ │
* * * * * command to execute
```

Figure 3.7 – Cron syntax (taken from https://en.wikipedia.org/wiki/Cron

For the reasons given above, the last solution was selected and integrated in the project.[. . .] These logs shown in Figure 3.8 show the good functioning of the system and the graph on Figure 3.9 further supports the argument.



```
Search logs    scheduledFunctionCrontab(europe-we…    All log levels    0 new logs

Time ↑         Level  Function          Event message
Jul 22, 2019
12:01:02.794 …        scheduledFunc…    Function execution started
12:01:03.108 …  i     scheduledFunc…    This will be run every day at 00:01 AM UTC!
12:01:04.880 …        scheduledFunc…    Function execution took 2087 ms, finished with status: 'ok'
Jul 23, 2019
12:01:02.016 …        scheduledFunc…    Function execution started
12:01:02.179 …  i     scheduledFunc…    This will be run every day at 00:01 AM UTC!
12:01:03.255 …        scheduledFunc…    Function execution took 1240 ms, finished with status: 'ok'
Jul 24, 2019
12:01:02.486 …        scheduledFunc…    Function execution started
12:01:02.673 …  i     scheduledFunc…    This will be run every day at 00:01 AM UTC!
12:01:04.029 …        scheduledFunc…    Function execution took 1545 ms, finished with status: 'ok'
```

Figure 3.8 – Scheduled function logs (taken from the Firebase console)

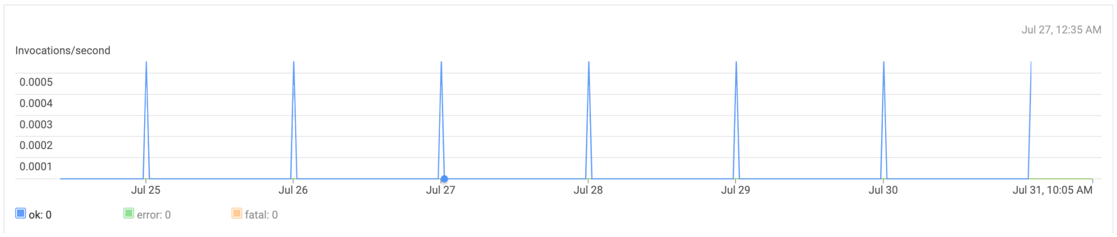


Figure 3.9 – Scheduled function graph (taken from the Google Cloud console)

3.3.5 verification code - scanning process

During the scanning process, a four-digit verification code is needed to fully validate the offer. This code needs to be entered by the shopkeeper on the user's phone once the verification alert is prompted (see Figure 3.10). This part is essential to avoid counterfeit. Indeed, a client can copy the behaviour of the application in order to make the staff of an establishment believe that the customer is entitled to a drink or an offer when this is not the case. For example, the client can make a scanning process that validates no matter the QR code. To enhance this verification part, the four-digit code given to the establishment are renewed each month with the help of a schedule function. This code is communicated by email to the owner of the establishment. But even after this extra step, the client can still cheat the process by making his application accepts all four-digit code. In fact, this problem can only be really solved if a check is made directly between the staff and the server, via a device in the store. For example, instead of establishment having QR code, the customers will have it in their phone. They present the code to the device which will scan and and validates only the good QR code. Because this process is very costly and hard to implement, it is left for a later iteration of the application.

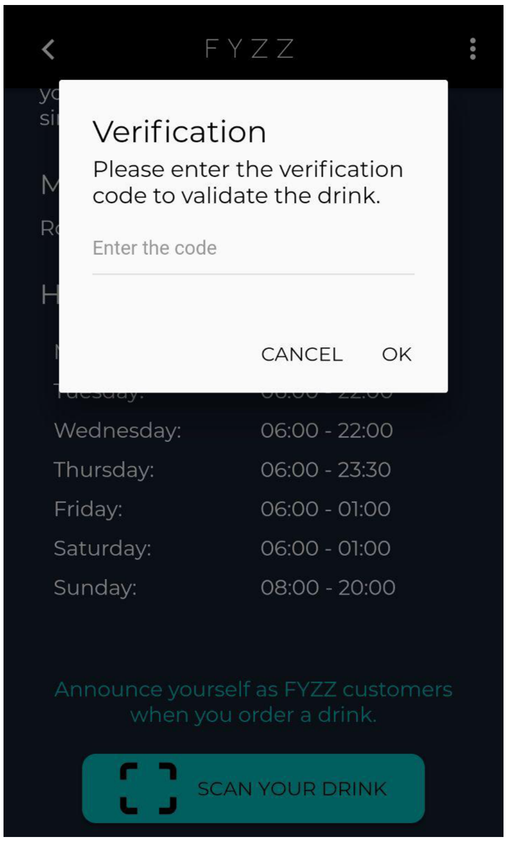


Figure 3.10 – Verification alert

4 Conclusion

4.1 Summary

To summarize in one sentence, Fyzz is an application build on iOS and Android that involves three type of stakeholders (FYZZ employees, owners of establishments and users) and uses a stack of several different modern technologies to provide a subscription-based service that is to offer a free drink per day and a commercial benefit. The challenge of this type of project is to build a simple yet userfriendly application to retain the users while providing a strong security to secure the data. Process like the scanning need to be thought in the user's direction which means the fewer clicks the better. To achieve this, a good database architecture is needed. In addition to that, and because FYZZ is modular and will probably facing changes in the near future, the type of database need to be modular as well. NoSQL over SQL structure was a good choice to address this concern. Finally, as FYZZ core business process evolves around the scanning process, a good solution for this transaction is required. Cloud functions that are scheduled and triggered at a specific time provide a good time efficiency in terms of coding time and are not complicated to integrate while giving a good amount of security as the code is stored on a remote server.

4.2 The future

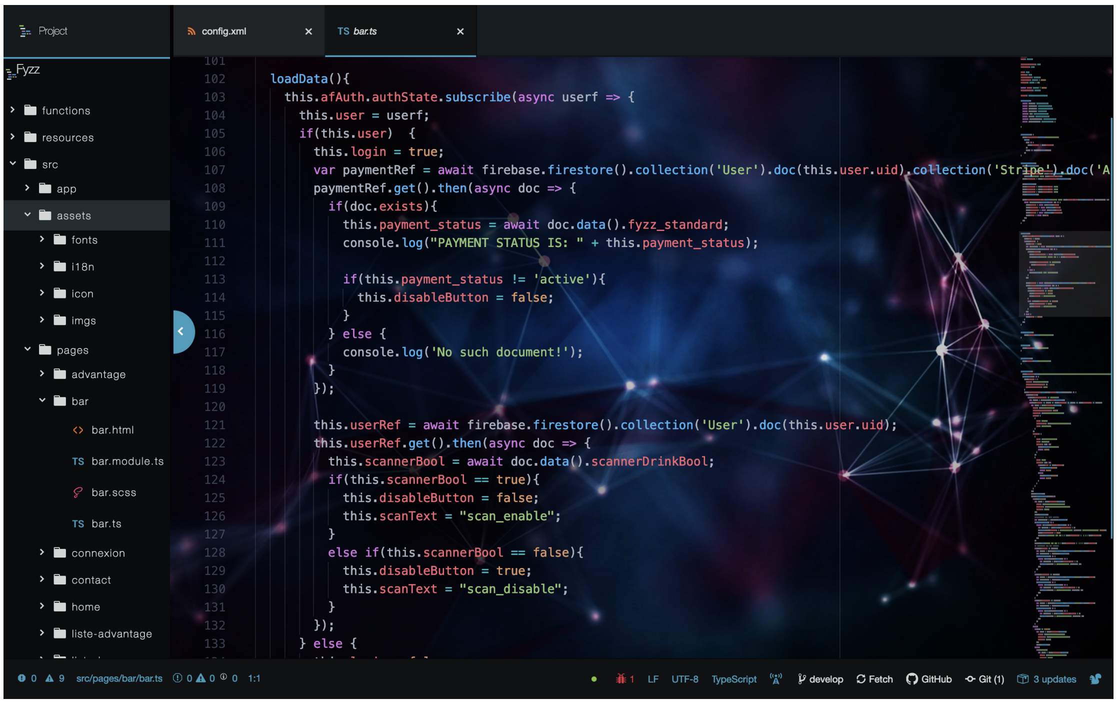
As stated above, FYZZ will probably face changes either in terms of business or programming aspect. More services such as the two already included (one drink and one promotion per day) can be added. A premium subscription can be considered and several functionalities can be integrated, such as the live direction on the map, or notification/vibration when the user is located less than five meters from an establishment. A new way of transaction can be implemented thus reducing the number of clicks when the user scans the offer. However, this requires to completely change the scanning process: instead of scanning a QR code specific to an establishment, a QR code is given to each users and a scanning device are mounted in each referenced establishments.

FYZZ welcomes changes as growth is only possible through challenging the mind to think new thoughts.

4.3 Personal challenge

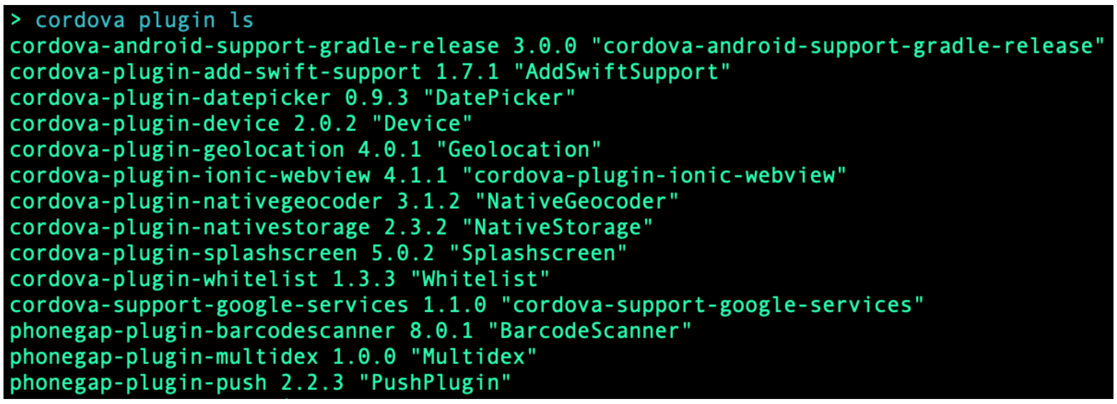
This work has been a challenge in several ways. First, finding a good application layout to enhance the user experience was not easy. As I am a computer science student and not a marketing student, I had to test different layouts through surveys to find the best one. Once the layout chosen, I had to face how to set up a good transaction process of how to validate the offer. As this handles sensitive data and avoiding the users to time cheat at all cost was my priority, I had to find a way of building a good database structure and a good solution to counteract the potential fraudsters. This project involving many different technologies requires a lot of involvement, concentration and motivation to implemented all the pieces of this puzzle.
Having the chance to work on a real application made available to the public has given me a lot on both a personal and professional level. Thanks to these security problems, I was able to develop new programming methods in software engineering.

A Appendix



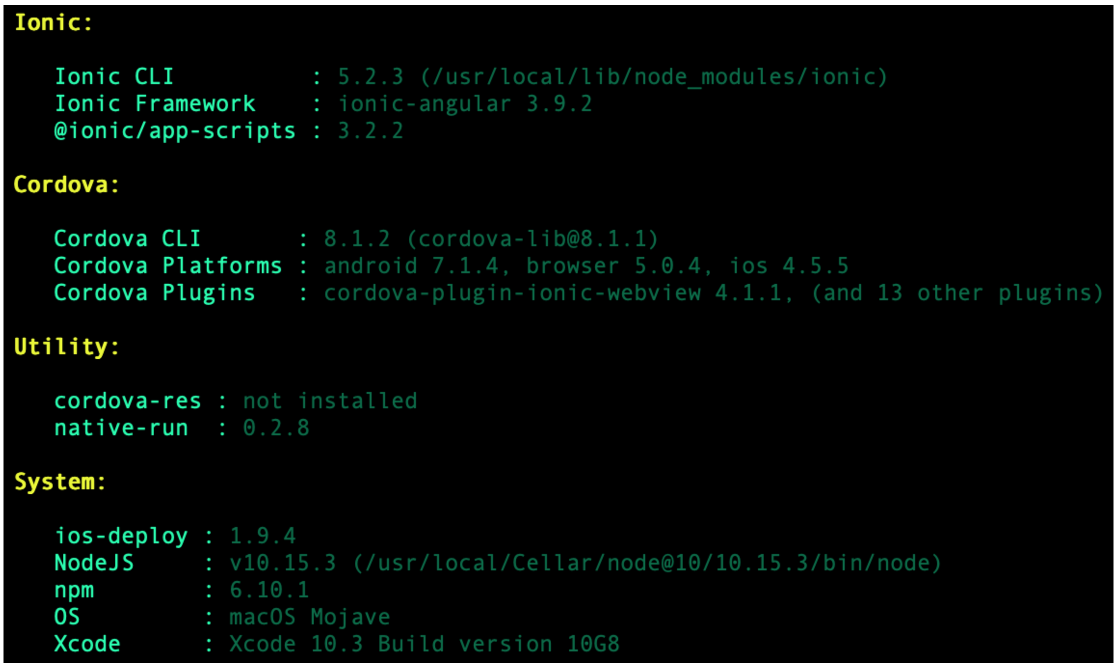
```
loadData(){
  this.afAuth.authState.subscribe(async userf => {
    this.user = userf;
    if(this.user)  {
      this.login = true;
      var paymentRef = await firebase.firestore().collection('User').doc(this.user.uid).collection('Stripe').doc('A
      paymentRef.get().then(async doc => {
        if(doc.exists){
          this.payment_status = await doc.data().fyzz_standard;
          console.log("PAYMENT STATUS IS: " + this.payment_status);

          if(this.payment_status != 'active'){
            this.disableButton = false;
          }
        } else {
          console.log('No such document!');
        }
      });

      this.userRef = await firebase.firestore().collection('User').doc(this.user.uid);
      this.userRef.get().then(async doc => {
        this.scannerBool = await doc.data().scannerDrinkBool;
        if(this.scannerBool == true){
          this.disableButton = false;
          this.scanText = "scan_enable";
        }
        else if(this.scannerBool == false){
          this.disableButton = true;
          this.scanText = "scan_disable";
        }
      });
    } else {
```

Figure A.1 – A screenshot of Atom text and code editor. The left panel shows how the files are organized. In a page folder (here the bar page), four files can be found: one html, two typescript and a scss file.



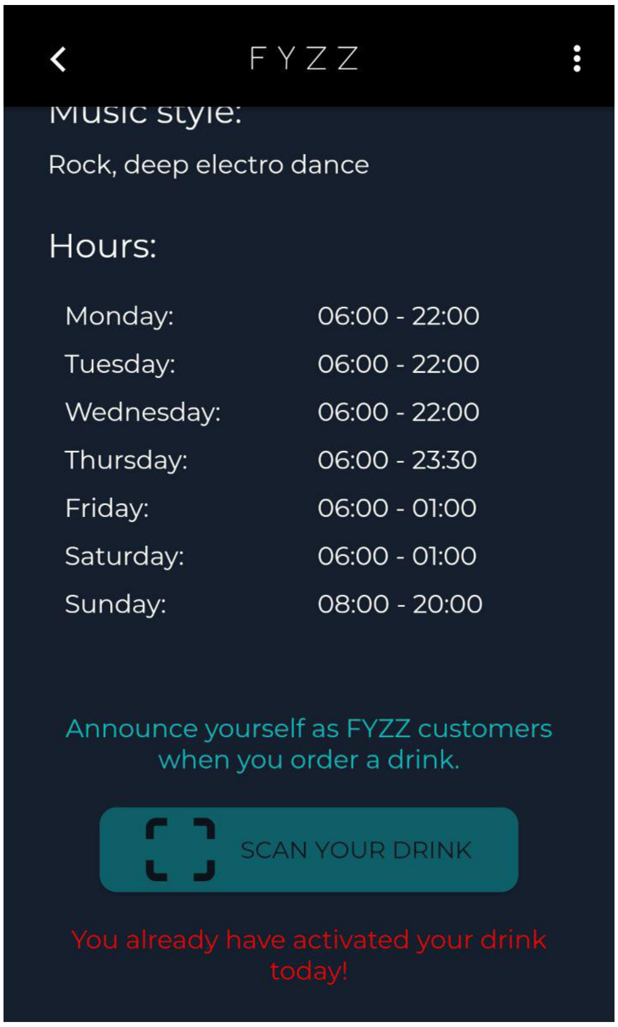
```
> cordova plugin ls
cordova-android-support-gradle-release 3.0.0 "cordova-android-support-gradle-release"
cordova-plugin-add-swift-support 1.7.1 "AddSwiftSupport"
cordova-plugin-datepicker 0.9.3 "DatePicker"
cordova-plugin-device 2.0.2 "Device"
cordova-plugin-geolocation 4.0.1 "Geolocation"
cordova-plugin-ionic-webview 4.1.1 "cordova-plugin-ionic-webview"
cordova-plugin-nativegeocoder 3.1.2 "NativeGeocoder"
cordova-plugin-nativestorage 2.3.2 "NativeStorage"
cordova-plugin-splashscreen 5.0.2 "Splashscreen"
cordova-plugin-whitelist 1.3.3 "Whitelist"
cordova-support-google-services 1.1.0 "cordova-support-google-services"
phonegap-plugin-barcodescanner 8.0.1 "BarcodeScanner"
phonegap-plugin-multidex 1.0.0 "Multidex"
phonegap-plugin-push 2.2.3 "PushPlugin"
```

Figure A.2 – Plugins used.



```
Ionic:

   Ionic CLI          : 5.2.3 (/usr/local/lib/node_modules/ionic)
   Ionic Framework    : ionic-angular 3.9.2
   @ionic/app-scripts : 3.2.2

Cordova:

   Cordova CLI       : 8.1.2 (cordova-lib@8.1.1)
   Cordova Platforms : android 7.1.4, browser 5.0.4, ios 4.5.5
   Cordova Plugins   : cordova-plugin-ionic-webview 4.1.1, (and 13 other plugins)

Utility:

   cordova-res : not installed
   native-run  : 0.2.8

System:

   ios-deploy : 1.9.4
   NodeJS     : v10.15.3 (/usr/local/Cellar/node@10/10.15.3/bin/node)
   npm        : 6.10.1
   OS         : macOS Mojave
   Xcode      : Xcode 10.3 Build version 10G8
```

Figure A.3 – Information setup about the project, the system and environment.



Music style:

Rock, deep electro dance

Hours:

Monday:	06:00 - 22:00
Tuesday:	06:00 - 22:00
Wednesday:	06:00 - 22:00
Thursday:	06:00 - 23:30
Friday:	06:00 - 01:00
Saturday:	06:00 - 01:00
Sunday:	08:00 - 20:00

Announce yourself as FYZZ customers when you order a drink.

You already have activated your drink today!

Figure A.4 – Used scan

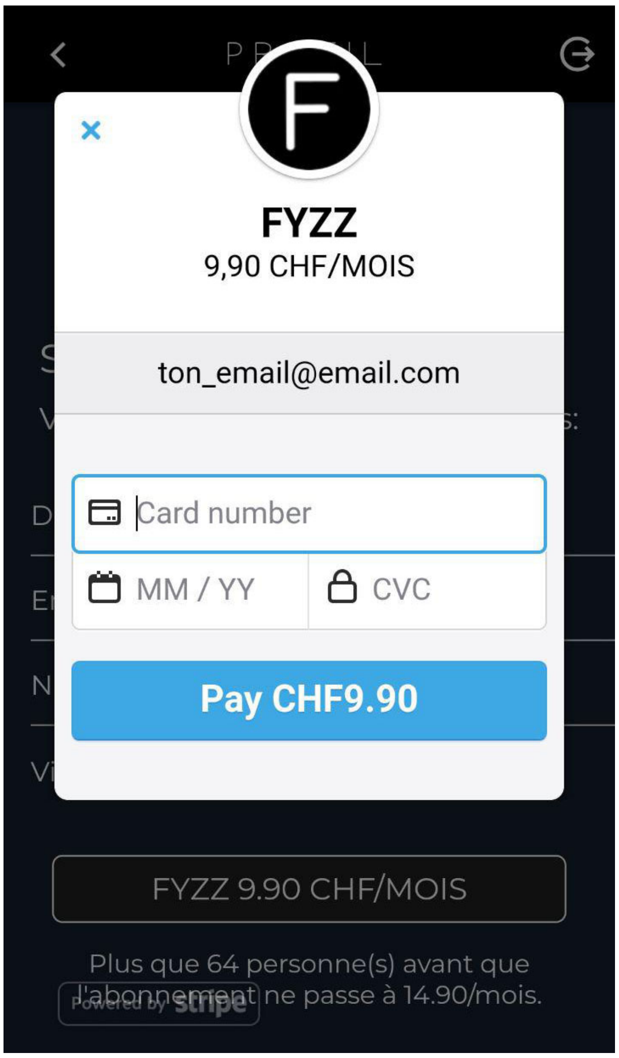


Figure A.5 – Stripe payment form

References

[1] APACHE CORDOVA, *Apache Cordova.* https://cordova.apache.org/. Visited on 28-06-2019.

[2] FIREBASE, *Cloud Firestore.* https://firebase.google.com/docs/firestore. Visited on 28-06-2019.

[3] GOOGLE, *Google Cloud Functions.* https://cloud.google.com/functions/. Visited on 12-07-2019.

[4] IONIC, *Ionic Framework.* https://ionicframework.com/. Visited on 28-06-2019.

[5] I. KANTOR, *Async/await.* https://javascript.info/async-await. Visited on 30-06-2019.

[6] E. LAMPRECHT, *The Difference Between UX and UI Design - A Layman's Guide.* https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/. Visited on 26-06-2019.

[7] A. MEIER AND M. KAUFMANN, *SQL- & NoSQL-Datenbanken*, Springer Vieweg, Fribourg, 1 ed., 2016.

[8] NODE.JS, *About Node.js.* https://nodejs.org/en/about/. Visited on 28-06-2019.

[9] NPM, INC., *About npm.* https://docs.npmjs.com/about-npm/. Visited on 28-06-2019.

[10] V. OKAFOR, *Working with Firebase Authentication.* https://medium.com/@valokafor/working-with-firebase-authentication-8f7dcb016e84. Visited on 01-07-2019.

[11] N. PARSON, *JavaScript: Promises and Why Async/Await Wins the Battle.* https://dzone.com/articles/javascript-promises-and-why-asyncawait-wins-the-ba. Visited on 07-08-2019.

[12] S. SRIVASTAV, *How Important is UI UX Design in an App Development Process?* https://appinventiv.com/blog/importance-of-ui-ux-design-during-your-mobile-app-development/. Visited on 26-06-2019.

[13] D. STEVENSON, *Why are the Firebase API asynchronous?* https://medium.com/google-developers/why-are-firebase-apis-asynchronous-callbacks-promises-tasks-e037a6654a93. Visited on 30-06-2019.

[14] STRIPE, *Security at Stripe*. https://stripe.com/docs/security/stripe. Visited on 03-07-2019.

[15] WIKIPEDIA CONTRIBUTORS, *Cron*. https://en.wikipedia.org/wiki/Cron. Visited on 12-07-2019.

[16] ———, *Trusted timestamping*. https://en.wikipedia.org/wiki/Trusted_timestamping. Visited on 12-07-2019.