

**FACULTY OF ENGINEERING OF THE
UNIVERSITY OF PORTO**

a.a. 2021/2022



**FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO**

Mobile Computing

Practical Assignment #1 / Design and Development

Order and pay Android app for an electronics shop

Antonio Gomès (202111218)

Claudio Savelli (202111375)

Eliott Tardieu (202111217)

ZAMBLAPP ELECTRONIC SHOP

I. INTRODUCTION

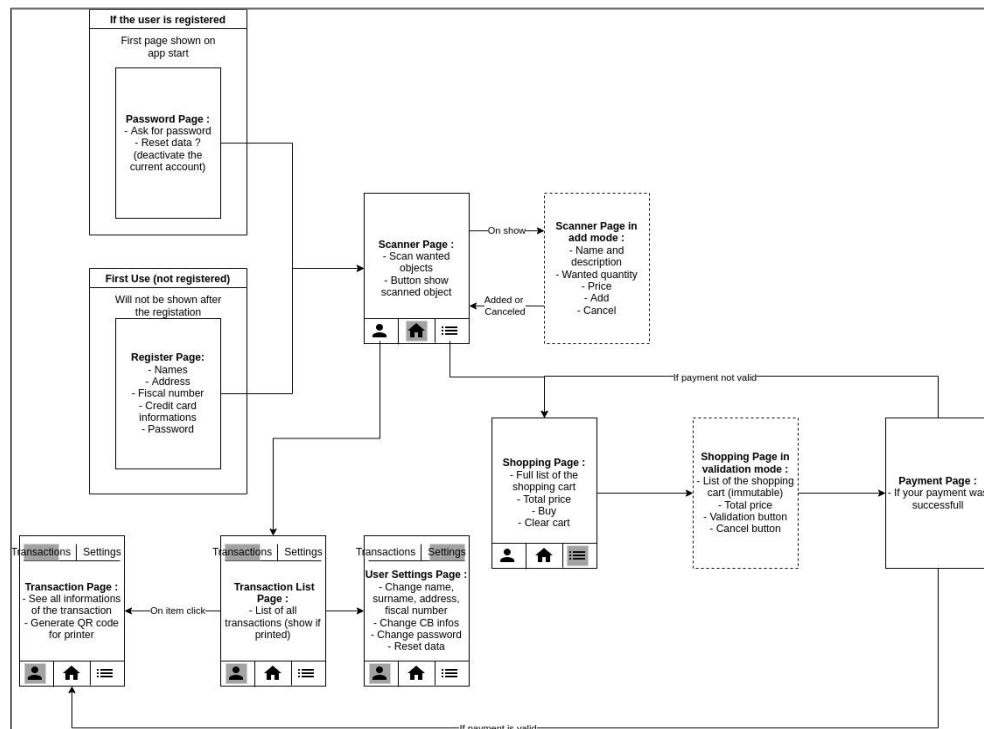
The purpose of this report is to allow the reader to follow the main points, the tests carried out and the code behind the application that we have developed step by step. In the introductory part, as was done in the development of the app, the design underlying the application will be shown and how the data structure for the server's operations has been organized with the description of the entities and tables.

The application was programmed using Kotlin language, the IDE used was AndroidStudio and the sharing (and versioning) of the work took place through GitHub.

Application Design

Through the application "Draw.io" we graphically defined the idea and the initial organization that we conceived for the application to try to organize logically and immediately for the user all the functions that we had to implement without the user getting lost inside it. The main goal was to make the interface as simple and clean as possible, with little text for each view.

The result obtained is attached:

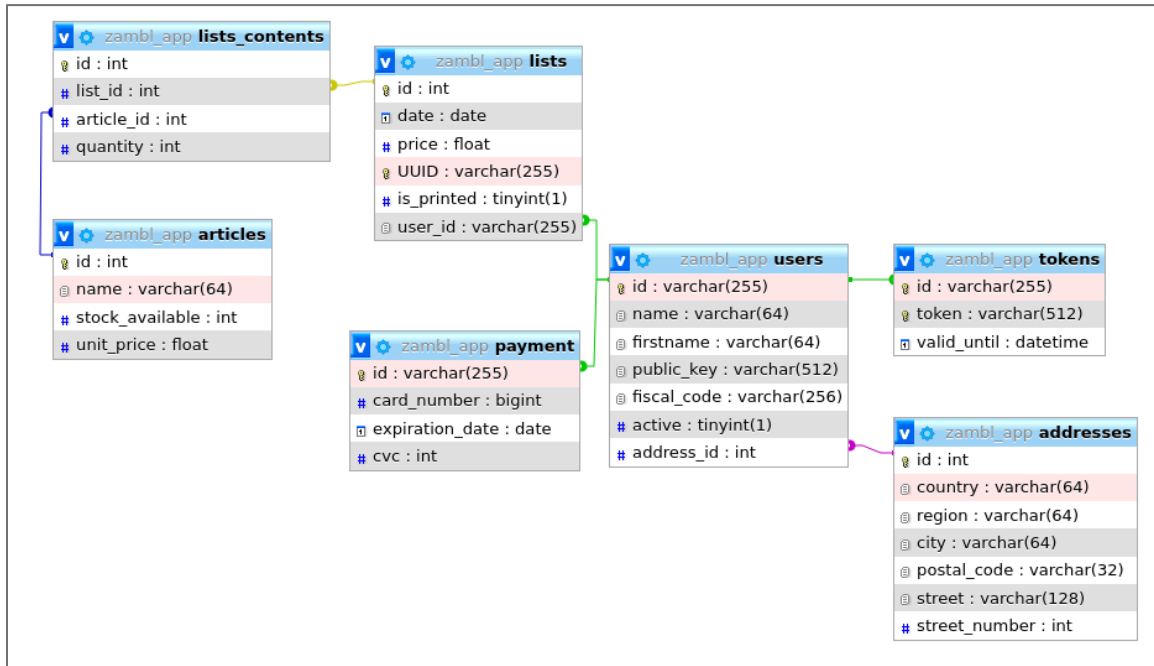


Server Design

The RESTful server was written in PHP language using the PHPStorm IDE.

Just as happened with the design of the application, before starting to build the server with its related functions we took care of structuring it adequately also to be able to make the writing of the actual code more immediate.

The organization of the database tables we have used is attached:



For the server, we used a framework homemade, similar to Laravel, with which Eliott already had experience. We followed the PDO approach by instantiating objects instead of manipulating data using MySQLi, we will come back to this more accurately in the Server part of the report.

II. SETUP

Application and Server Setup

Run the application in the last version of AndroidStudio (the minimum API required is 26 for the communication with the server). We have made a custom docker container to host the server locally if you don't have ubuntu with apache2 and MySQL server installed. You may follow the different READMEs to set up the project and the server properly.

Generating QR Codes

To use the QR function within the application, 15 different articles have been entered into the server database (see the db_sample.sql file on the server). To be able to "scan" them it will be necessary to generate the QR code online for the ID of the object (e.g., 5 -> MacBook 12 because MacBook 12 has ID = 5 in the database).

To generate the QR code it is, therefore, necessary to go to any site that generates QR codes (e.g., <https://4qr.com/#text>) and enter a text with the relative number.

Example of generated QR Codes is attached:



iPhone 12



Smartwatch 2



MacBook 12

III. APPLICATION

Introduction

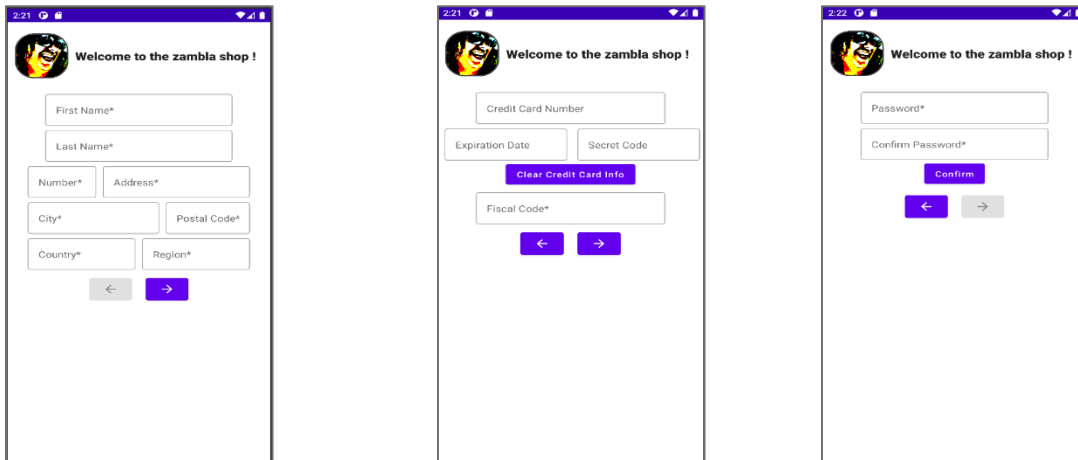
The goal during the development of the application was to create easily readable, intuitive code, taking full advantage of the various libraries offered by Kotlin and AndroidStudio.

To begin with, the most important and most used, which certainly had a greater impact on the development of the application was Jetpack Compose. Jetpack Compose *"is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs."* [1]. Thanks to this tool, we could lighten the code and have clean, tidy, and easy to view screens.

Registration/Login page

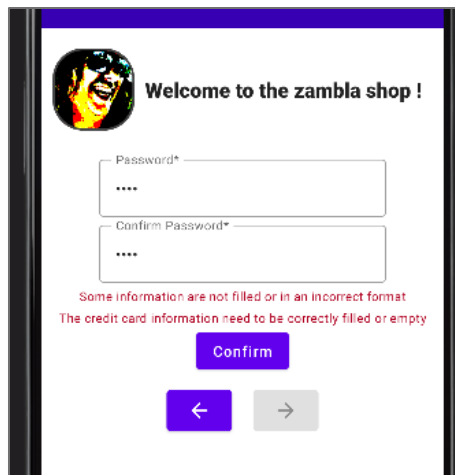
The registration page is divided into three views, to create it a mechanism of buttons and image changes was used in which by clicking on the next or back button we go to change the numerical value of the counter and depending on the value we have it will be shown within our screen a different text.

The registration page of the application is attached:



Furthermore, in the event of errors in filling in the mandatory fields, the application will block the registration by requesting to modify the aforementioned fields so that they are valid. The fields are scanned by regexes to make sure they are correctly filled. Also, the user may not want to give payment information while registering, we made this possible, but then the payment will fail. The user may then edit his account later, to enter a valid payment method.

An example of the error page is attached:



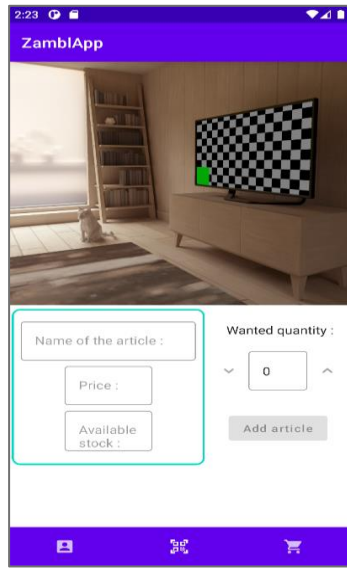
In the application

Once you have entered the application the views are almost the same that we had initially entered in the design and evaluation phase.

The reading of the barcode and the insertion of the desired products in the cart before making the purchase deserve particular attention. For the barcode reading, we decided to use the following library: MLKit made by Google. We used the barcode scanning feature. MLKit *“brings Google’s machine learning expertise to mobile developers in a powerful and easy-to-use package. Make your iOS and Android apps more engaging, personalized, and helpful with solutions that are optimized to run on device”* [2], so it allows

you to exploit some means offered by Google to be able to intelligently read some images through the camera.

The view obtained is attached:



How is possible to see from the screenshot, that within the application, we don't just read and automatically insert the read code, but we automatically update the viewer on the left which will not only tell us the name of the item and the price, but also the quantity available in the warehouse. Every time the user logs in, he receives a temporary token and a list containing the name and current quantity in stock of each registered object.

As mentioned before, the other screens in the application are standard and do not require particular explanations, being the same ones present in the image of the initial project. However, below it is possible to find some features that we have decided to include even if not expressly requested, the aim of which is to make the user more engaged in using it.

A feature we added is the possibility for the user to modify some of his data (e.g., address in case he changes it and/or credit card in case it should expire). We made this because we wanted to give the possibility for the user to be able to register without immediately entering his card details and then being able to enter them, if desired, at a later time.

Another feature that has been included even if not expressly requested is the presence of loading screens; in fact, since there are numerous communications with the server during which the user not only cannot do anything but must wait for a response from it, we thought that inserting this screen rather than just blocking the application could make it a more dynamic and less boring experience.

IV. SECURITY

Given the nature of the application and the need to record sensitive informations such as the credit card data of the customer, it was necessary to have a particular focus on the security to be ensured within the application and with the communication with the server, so much so that it deserves a specific paragraph within the report.

Registration and Login

At the time of registration, the customer enters his private data, which is saved in the cache and the server database. Once the registration has been made (which is single for each device, therefore the same account cannot access from different devices, but it is necessary to create a new one for each of these), the initial screen of the application will change permanently until the same account is deleted or account. From this moment on, the new initial screen will be an initial login screen in which the password is saved only within the phone and not on the server (therefore it is extremely difficult to be able to enter another account, having to have available not only the electronic device of the but also the password).

Server communication

When creating a new account, the application generates a public key and a private key for communication with the server and sends the first to the server which will associate it with the unique account code (UID). Carrying out all communication of transactions with the server through this method alone is very expensive in algorithmic terms and creates problems with encrypted communication with printers (considering that we chose not to register the printers in the database, it was then impossible to have a secure connection that could resist a Man in the Middle kind of attack).

A method widely used in this kind of communication solves this problem: temporary tokens. At the time of registration and every time the user logs in, the server will generate a temporary token that is sent to the user encrypted with his public key. Once the user receives it, he decrypts it with his private key, thus obtaining the plaintext token that will be used for communication in that time window. The token is stored in the user's SharedPreferences. Each temporary token has an expiration delay of 24 hours, as it may take time for the customer to go around the shop, fill the cart and then make the payment.

Printer and Security

In this way, user-printer and printer-server communication are also made easier and safer since the printer will only act as an intermediary between the user and the server without having to perform operations on requests and without ever storing them in its memory. The printer reads a QR code containing the information and transforms them into JSON to do a verification query to the server. The server will then tell the printer if the list is valid or not if it was made by the user owning the token, and if everything goes right, the printer will show the list's information on its screen.

Encryption Controller

In the application, the encryption controller takes care of the following operations: In particular, the pair of keys is generated and inserted into the KeyStore with its alias linked to the name of the application, in this way we do not go to save the keys directly into the memory of the device (dangerous and weak from any kind of attacks), but within a system that makes it impossible to acquire the key if the request is not made by the application. In addition, the key always remains invisible during the operation and is never loaded into the cache. On the other hand, the temporary token, given its nature, can be saved in memory as a private entity without causing security problems.

An example of basic user-server communication is the following:

- i. The client makes a request showing who it is through the private key signature method (asymmetric key) and thus obtains a temporary token from the server, which saves the generated token and then attaches it to the user to whom it sent it.
- ii. The user, after having obtained and decrypted the token, uses it as a “secret” key (symmetric key), in order to encrypt and decrypt the messages exchanged with the server, which will easily be able to recognize the user who sent the message having saved the temporary token in the database and being able to trace back to the applicant, starting from the latter.
- iii. The token every 24 hours (or every new login) is updated and replaced with the previous one, this allows us not to risk being subjected to any brute force attacks (even if their chances of succeeding are very small as we are using UUID v4 for tokens).

Encryption Algorithms

The symmetric key algorithm is when the key is unique and secret, this way only the two communicating members are aware of the key. The public key algorithm works as a sort of cipher, so the key is used both to encrypt and decrypt the text. Instead, we talk about an asymmetric key algorithm when we have two keys available, one public, which can be seen by everyone, and one private, known only by the owner of the key itself (i.e., by whoever sends the message). With this method, we know that a text encrypted with the public key can only be decrypted with the private key and vice versa. This method is very useful and used for digital signatures, or to have secure communication in general. Usually, for reasons of time (it requires a greater number of calculations to decrypt with the asymmetric key), as also happens in this case at the beginning of the communication, the cypher (the private key) is exchanged to carry out the secure communication in that time frame, after which once the communication is finished the cypher is thrown away.

The asymmetric key encoding and decoding in the application use the "RSA/ECB/PKCS1Padding" encryption method. RSA (Rivest-Shamir-Adleman) is an asymmetric key encryption method widely used nowadays, while ECB (Electronic Code Block) is the simplest block encryption method we have, which is no longer used since it goes to encode the same messages in the same way. In our case, we did not worry about this problem so much as we use this type of encoding only to exchange the temporary token generated randomly every new login. This particular type of encoding/decoding is already automatically implemented in the Java libraries, which is why we have decided to use it.

Symmetric key encryption and decryption use the "AES/CBC/PKCS5Padding" method instead. AES (advanced encryption standard) is the method used to encrypt data, while CBC stands for Cipher-Block chaining, which is the successor of the ECB. In fact, with this type of encryption, if the CBC method is not used, we will have those two identical texts once encrypted is still the same, and this leads us to have some security weaknesses because starting from this knowledge a potential attacker can have a variety of information available simply by seeing that the same message is sent multiple times (he could try a plain-text attack for example). In this case, it's important to use CBC instead of ECB because we're going to send every kind of communication user/server using the token (as the Symmetric key encryption). To avoid this, an IV (Initialization Vector) is used, which allows us to always have different encryption depending on the IV that is generated every time a message is sent. As in the previous case, this encoding/decoding is already automatically implemented in the Java libraries, which is why we have decided to use it.

V. SERVER

Server Communication

For communication with the server during the various tests carried out in creating the application (Normal HTTP communication, Ktor) we decided to use Volley to make the code as simple and clear as possible. Volley is an HTTP library that does networking for Android Apps easier, cleaner, and faster. Volley allows you to implement a listener pattern and then serialize the data obtained to use them for our application. Once communication has been made with the server through Volley, Gson was used for JSON handling. Gson is "a Java serialization/deserialization library to convert Java (and Kotlin) Objects into JSON and back" [4]. Furthermore, for the communication with the server, all the models of the possible requests and responses that can be received from the server have been inserted into the models/server folder in the application to be able to serialize them easily.

Server Architecture

For the server's architecture, we applied a psr4 with FastRoute pattern, meaning simple clear Routes and namespaces. At first, we implemented MySQLi to do simple requests to our MySQL server, but then we decided to go for the PDO pattern by instantiating the models from the database. This way, we could easily handle login, register, and any other queries.

As for the security, we used the OpenSSL library to encrypt and decrypt both symmetric and asymmetric data. We generate UUID v4 for Tokens, User ID, and List UUID. As for the requests that are sensitive (such as address or payment modification), we use an integrity hash verification to make sure that the encrypted data integrity is valid. For this, we do a hash of the user ID concatenated with the content in JSON format. Thanks to the token, we know which user is sending the data, so we decrypt the data using the token and we then hash the concatenation of the user ID and the content. This allows us to prevent ourselves from a Man in the Middle kind of attack, which could edit the data sent.

VI. PRINTER

How does it work

The main goal of the printer is to read a QR code (code already written for reading the product inside the store) and to then verify (by doing a server query) that the list is valid, not yet printed, and was paid by the correct user. If so, it prints the list content, date, and price (it displays it on the screen as a simulation of printing), if not it shows an error message.

As already mentioned, thanks to the system implemented for security, the printer will only act as an intermediary between server and client without going to modify the message sent and without even being able to read it (not knowing how to decrypt it, because only the server knows).

VII. TESTS

Application Tests

After creating our application, we are committed to creating lists, reading QR codes and trying to change the number of items in our cart or at the time of selection.

In addition, various tests were carried out to create, delete, connect, and modify the profile/credit card.

Security Tests

We tested both Asymmetric and Symmetric encryption and decryption, making sure the public & private keys were persistent.

After carrying out the previous tests inside the application, we took care to carry out the same ones but this time taking into consideration the server, to send and use the user's public key correctly, to transform the public key with the following line of code:

“android.util.Base64.encodeToString(publicKey.encoded, android.util.Base64.NO_WRAP)”

to allow the server to use it to decrypt messages. Once we had ensured that they were sent to the server in the desired way, we were able to proceed with encryption and decryption with messages sent/received by the server itself and no longer working exclusively within our application.

Server Communication Test

This type of test is the one that took us the most time, since we not only had to test the communication with the server itself with attached encryption and decryption of the data sent, but also that the server correctly read the message, the relative affected tables were modified (e.g. purchase of a product causes the quantity of the same in the warehouse to decrease), and solving issues between the different encryption libraries used in the server and the application.

We also paid particular attention to error handling and informative messages (status messages), which allowed us to debug the communication more efficiently and will enable the application, through informative messages, to know why the transaction or communication was unsuccessful.

Printer Test

Once the exact client-server communication was tested, the test with the printer was immediate, since its only task, as already described above, is simply to act as an intermediary in the communications between the two main bodies.

VIII. CONCLUSIONS

Considerations

The development of the following application has let us not only learn how to use and know all the basic mechanisms of mobile computing but also allowed us to learn more about a programming language almost unknown to us (Kotlin). In addition, the use of numerous libraries and internet research has permitted us to know and learn how to use numerous useful tools not only for mobile programming but in general for all types of programming.

The creation of the RESTful server was a very first experience for us, and while it's not the main application in this project, it was a very nice experience to implement an API.

The teamwork and the subdivision of the project into 3 distinct models allowed us to deepen the sense of teamwork within a project for the creation of software, not only at the communicative and organizational level but also with the use of specific software for shared programming (GitHub).

BIBLIOGRAPHY

- [1] Build better apps faster with Jetpack Compose (<https://developer.android.com/jetpack/compose>)
- [2] Machine learning for mobile developers (<https://developers.google.com/ml-kit>)
- [3] Volley overview (<https://google.github.io/volley/>)
- [4] Gson overview (<https://github.com/google/gson>)