U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

The Acme Supermarket

Mobile Computation 2022/2023

Made by:

Joana Mesquita - up201907878

Alberto Cunha - up201906325

João Costa - up201907335

1. Introduction

This report aims to describe the first assignment of the Mobile Computation course.

The assignment consisted of creating two apps for a supermarket, The Acme Supermarket. These apps are meant to allow customers to experience a new, fully automated, way of shopping. Instead of the usual Cashier handling the customer's payment, the customer would be able to perform that task independently cutting on waiting time to check out, by scanning the QR codes labels of products they wish to buy and then, when it's time for checkout, by showing a QR code generated from their shopping list to the supermarket terminal, that will, after a successful purchase, open the gates to exit. This whole implementation was meant to be made for Android and with a REST service.

2. Architecture and Design

The project consists of three main services with another one created by us to aid in testing.

2.1. ACME

This app created in native android using Kotlin is the one that will be installed in costumers' devices to allow them to do their shopping independently with the capacity to both scan product labels and checkout at the exit. It also provides the user with other useful features such as a loyalty program to acquire discounts and the ability to retrieve the user's purchase historic. To protect users' information and money, the app encrypts and sign data when sending to the server, guaranteeing safety.

2.2. ACME Terminal

This app created in native android using Kotlin is installed in the supermarket's terminals and scans the codes generated by the customers' apps telling the customer if their purchase was completed successfully or not.

2.3. ACME Backend

This service was created using Spring Boot and corresponds to the server that communicates with the applications described in 2.1 and 2.2 and keeps the database with all the information of the supermarket's clients. It is not directly used by customers, rather it is used by the applications to perform actions that require acquiring or saving information (permanent storage). The server is also responsible for verifying signatures and decrypting messages, to protect the information of the users.

2.4. ACME QR code generator

For ease of testing, we also created another app using native android and Kotlin that simply creates QR codes for product labels.

3. Data Schemas

3.1. ACME

- **Data Classes**

Product – Date structure used when scanning products

```
data class Product(
    var uuid: String,
    var name:String,
    var price: Float
)
```

ProductAmount – Date structure used when receiving receipts and when getting products from the shopping cart.

```
data class ProductAmount(
    var uuid: String?,
    var amount: Int,
    var name:String?,
    var price: Float
)
```

Receipt – Date structure used when receiving receipts.

```
data class Receipt(
    var date: String,
    var price: Float,
    var items: ArrayList<ProductAmount>,
    var voucher: String

)
```

Voucher – Date structure used when receiving a voucher.

```
data class Voucher(
    var emitted: Boolean,
    var used: Boolean,
    var date: String,
    var uuid: String
)
```

VouchersInfo – Date structure used when receiving vouchers and the amount until the user receives their next voucher.

```
data class VouchersInfo(
    var vouchers: ArrayList<Voucher>,
    var valueToNext: Float
)
```

- **Shared Preferences**

User Information

The user info and the servers public key received when the user registers into the app are stored in the app's the shared preferences' fille "user_info" under the following keys:

- uuid - user's UUID
- name - user's name
- username - user's username
- server_public_key - user's UUID
- discount - amount of accumulated money by the user from using coupons
- total - total amount of money spent by the user

Shopping Cart

The products in the shopping cart are stored in the following shared preferences' filles with the product uuid as the key:

- "shopping_cart_prod_names" - has the product name
- "shopping_cart_prod_prices" - has the product price
- "shopping_cart_prod_amount" - has the product amount

3.2. ACME Terminal

The ACME Terminal simply receives the string from the QR code generated by ACM and passes it to the backend service without having any proper data structures.

3.3. ACME Backend

We used the following tables to store the information necessary for this app:

User – Saves information about the users of the app.

```
@Entity
@Table(name = "app_user")
public class AppUser {
    @Id
    @SequenceGenerator(
            name = "student_sequence",
            sequenceName = "student_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "student_sequence"
    )
    @Column(name = "app_user_id")
    private Long id;
    private String name;
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(unique = true)
    private String public_key;
    @Column(unique = true)
    private Long card_number;
    @Column(unique = true)
    private String uuid;
    private Float discount;
    private Float total;
    @OneToMany(mappedBy = "user")
    private Set<AppPurchase> purchases = new HashSet<>();
    @OneToMany(mappedBy = "user")
    private Set<AppVoucher> vouchers = new HashSet<>();
```

Product - Saves information about the products available at the supermarket.

```
@Entity
@Table(name = "product")
public class AppProduct {
    @Id
    @SequenceGenerator(
        name = "product_sequence",
        sequenceName = "product_sequence",
        allocationSize = 1
    )
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "product_sequence"
    )
    @Column(name = "product_id")
    private Long id;
    private String name;
    private Float price;
    @Column(unique = true)
    private String uuid;
    @OneToMany(mappedBy = "product")
    private Set<AppItem> items = new HashSet<>();
```

Purchase - Saves information about purchases made by user.

```
@Entity
@Table(name = "purchase")
public class AppPurchase {
    @Id
    @SequenceGenerator(
        name = "purchase_sequence",
        sequenceName = "purchase_sequence",
        allocationSize = 1
    )
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "purchase_sequence"
    )
    @Column(name = "purchase_id")
    private Long id;
    private Float total_price;
    private Date date;
    private Boolean emitted;
    @ManyToOne
    @JoinColumn(name = "app_user_id")
    private AppUser user;
    @OneToMany(mappedBy = "purchase")
    private Set<AppItem> items = new HashSet<>();
    @OneToOne
    @JoinColumn(name="voucher_id")
    private AppVoucher voucher;
```

Item - Necessary table to save the products associated with a purchase.

```
@Entity
@Table(name = "item")
public class AppItem {
    @Id
    @SequenceGenerator(
        name = "item_sequence",
        sequenceName = "item_sequence",
        allocationSize = 1
    )
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "item_sequence"
    )
    @Column(name = "item_id")
    private Long id;
    private Integer quantity;
    @ManyToOne
    @JoinColumn(name = "product_id")
    private AppProduct product;
    @ManyToOne
    @JoinColumn(name = "purchase_id")
    private AppPurchase purchase;
```

Voucher – Saves information about vouchers issued and their use.

```
@Entity
@Table(name = "voucher")
public class AppVoucher {
    @Id
    @SequenceGenerator(
        name = "voucher_sequence",
        sequenceName = "voucher_sequence",
        allocationSize = 1
    )
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "voucher_sequence"
    )
    @Column(name = "voucher_id")
    private Long id;
    private Boolean emitted;
    private Boolean used;
    private Date date;
    @Column(unique = true)
    private String uuid;
    @ManyToOne
    @JoinColumn(name = "app_user_id")
    private AppUser user;
    @OneToOne(mappedBy = "voucher")
    private AppPurchase purchase;
```

We add to create various data schemes to use for the request and response bodies. We highlighted some to show here:

NewPurchase – Request body for the endpoint that creates new Purchases.

```
public class NewPurchase {
    public List<ProductAndQuantity> products;
    public String user_id;
    public Boolean discount;
    public Optional<String> voucher_id;
```

ReturnNewUser – Response body for the endpoint that creates new Users. Returns the uuid of the new user and the public key of the supermarket.

```
public class ReturnNewUser {

    public String uuid;
    public String public_key;
```

SignedId – Request body for various endpoints. Example of a type of request body we used when the server needs to verify signatures.

```
public class SignedId {
    public String uuid;
    public String signature;
}
```

Encrypt – Request body for the endpoint that creates new products. The string is encrypted by the user and decrypted in the server.

```
public class Encrypt {
    public String encryption;
}
```

3.4. ACME QR code generator

Due to limitations in the number of characters that can be encrypted with the encryption algorithm used the product information is put in the following format before being encrypted:

product_uuid:product_name:product_price

4. Included features

4.1. ACME

- Create an account
- Log into your account
- Change password
- Change Credit Card number
- Scan product labels
- Add products to your shopping cart
- Remove products from your shopping cart or reduce their quantity
- Checkout with different checkout options
- See available vouchers
- See past receipts.

4.2. ACME Terminal

- Scan QR codes from customers
- Finish checkout transaction

4.3. ACME QR code generator

- Create valid, correctly encrypted QR codes

5. Performed tests

5.1. Backend

We tested each endpoint with Thunder Client to evaluate if they were working correctly. This includes not just testing if the REST API was receiving and responding correctly, but also the request handlers were functioning right. These things included things like database requests, cryptography, and string formatting.

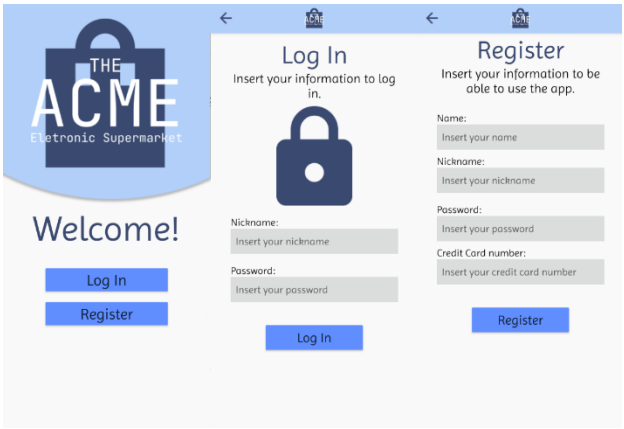
5.2. Integration

When we integrated the backend into the app, we tested by trying different inputs to see if the app would break. This was followed by the respective handling of error that would occur. These tests also include testing with various QR codes, including invalid ones.

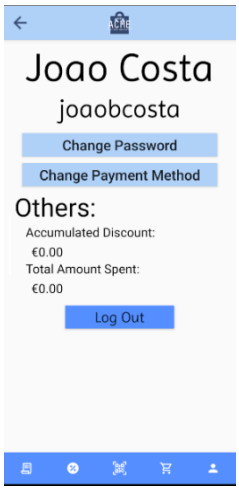
6. Way of use

6.1. ACME

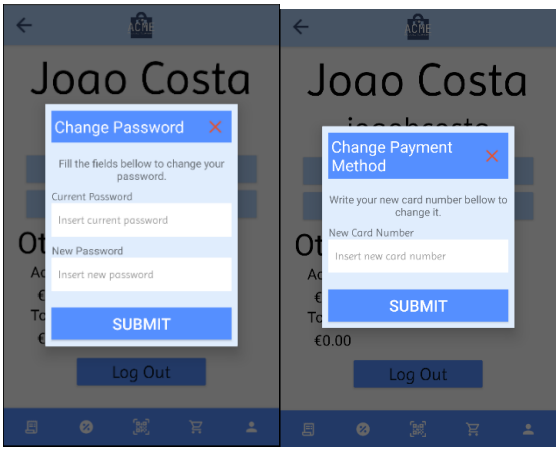
When the user first opens the application, they are presented with the home screen and can either login or register. If it is their first time using the application, they must first register.



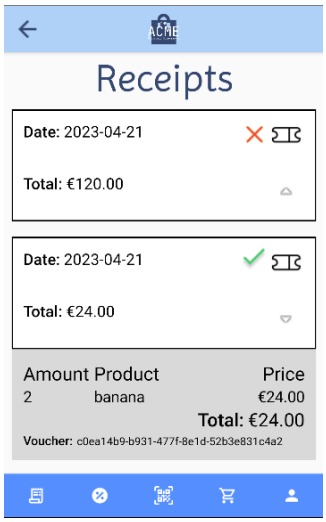
Once they login they will be automatically presented with the user profile, where they can see their name, nickname, total amount spent and accumulated discount.



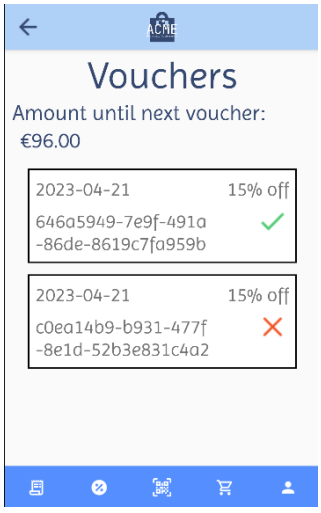
They can also change their password and payment method, which will both open dialog boxes to do so. From this page, using the navbar in the bottom, the user can navigate to all the features of the app.



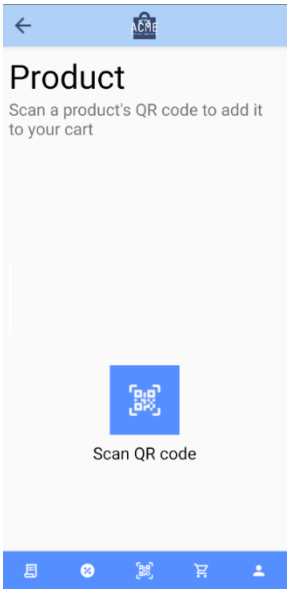
If the user clicks the receipts button in the navbar they will be redirected to the Receipts activity, in which the user can see the receipts from past transactions. For each receipt they can visualize the date of purchase, the amount spent, whether they used a voucher or not and can expand it for more details, such as the products that were purchased and what voucher was used.



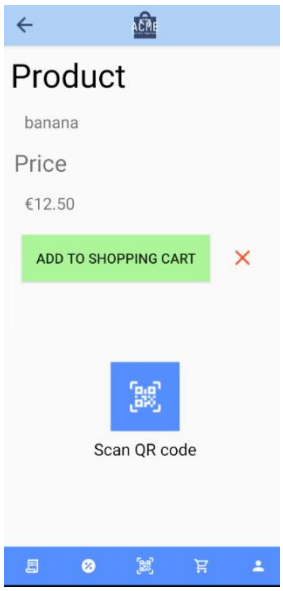
If the user clicks the vouchers button in the navbar they will be redirected to the Vouchers activity where they can see their vouchers and information about each one – their date, code, discount percentage and whether they have been used or not.



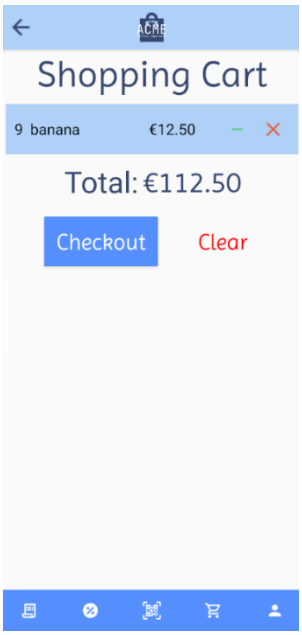
If the user clicks the QR code button in the navbar they will be redirected to the QR Code Scanner activity where they can scan for new products which will be added to the shopping cart.



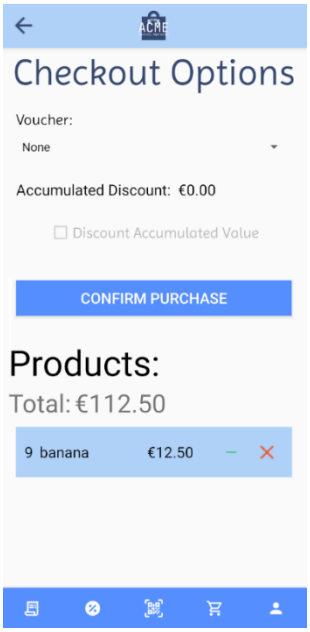
Once the scan is successful, the user can add the product to the shopping cart or opt to cancel the addition.



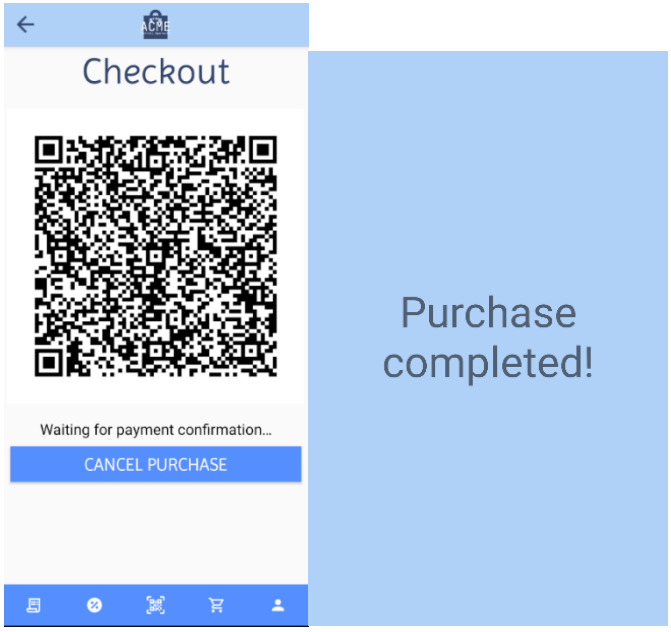
If the user clicks the Shopping Cart button in the navbar they will be redirected to the Shopping Cart activity in which they can see all the products currently added and remove them if needed. In the end they can clear the list of products or proceed to the checkout.



If the user clicks the checkout button in the shopping cart, they will be redirected to the Checkout Options activity, in which they can set what voucher they want to use, if they want to use one, whether they want to use the accumulated discount in this purchase, view the list of items from the current purchase and confirm the checkout.

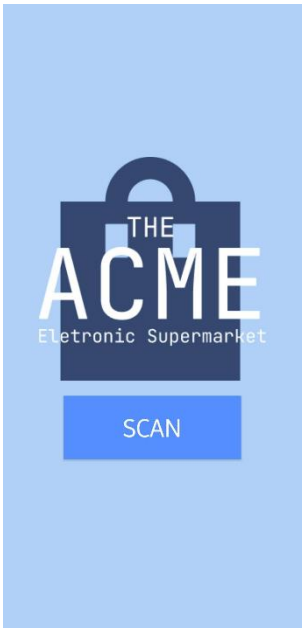


If the user confirms they will be redirected to a screen in which there is simply a QR code containing all the information relative to their purchase, such as the items and voucher used. This QR code must be scanned in the supermarket's scanners to complete the purchase and allow the user to leave the store.

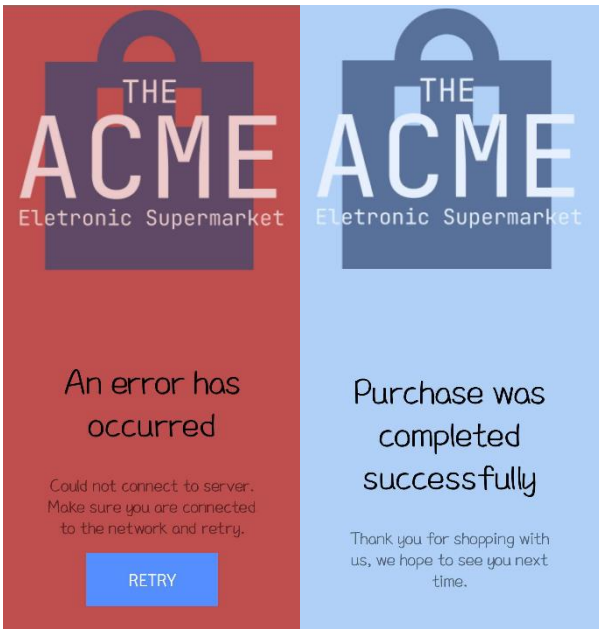


6.2. ACME Terminal

The terminal app has a main screen with a button to scan the checkout QR code.



After scanning the terminal app will show you if your purchase was completed correctly or not.



6.3. ACME QR generator

In the main screen you can insert the product name and price of the product the QR code is for.

Afterwards the app will generate a valid QR code.

7. How to run

Since the ACME Backend is not running on an online server, some steps are necessary to use the applications made.

First, the ACME Backend must be running with a Postgres database named acme_be with user postgres and password 123 on port 5432.

Furthermore, if the client and terminal are being used on emulators, then the BASE_ADDRESS constant on the class Constants in both apps must be “10.0.2.2”. If it’s being used in real phones connected to the computer through a USB connection, then the BASE_ADDRESS must be “localhost”, for this to work you might have to forward the port 8080 on your computer.

The QR code generator works independently of the other apps and can be run anywhere with no changes.

8. Conclusion

In conclusion, we successfully developed a supermarket application for ACME supermarkets, that allows users to do their shopping in a much more efficient way, by providing them with a system that relies on QR codes to keep track of the current purchase. Android was paramount in this implementation, since we used a comprehensive combination of its features to deliver all the features that were requested, such as security and QR code scanning.