Ticket and Cafeteria Ordering System

Table of Contents

Group Composition

Name	Email	GitHub
Carlos Veríssimo	up201907716@up.pt	carlosverissimo3001
Nuno Jesus	up201905477@up.pt	Nuno-Jesus

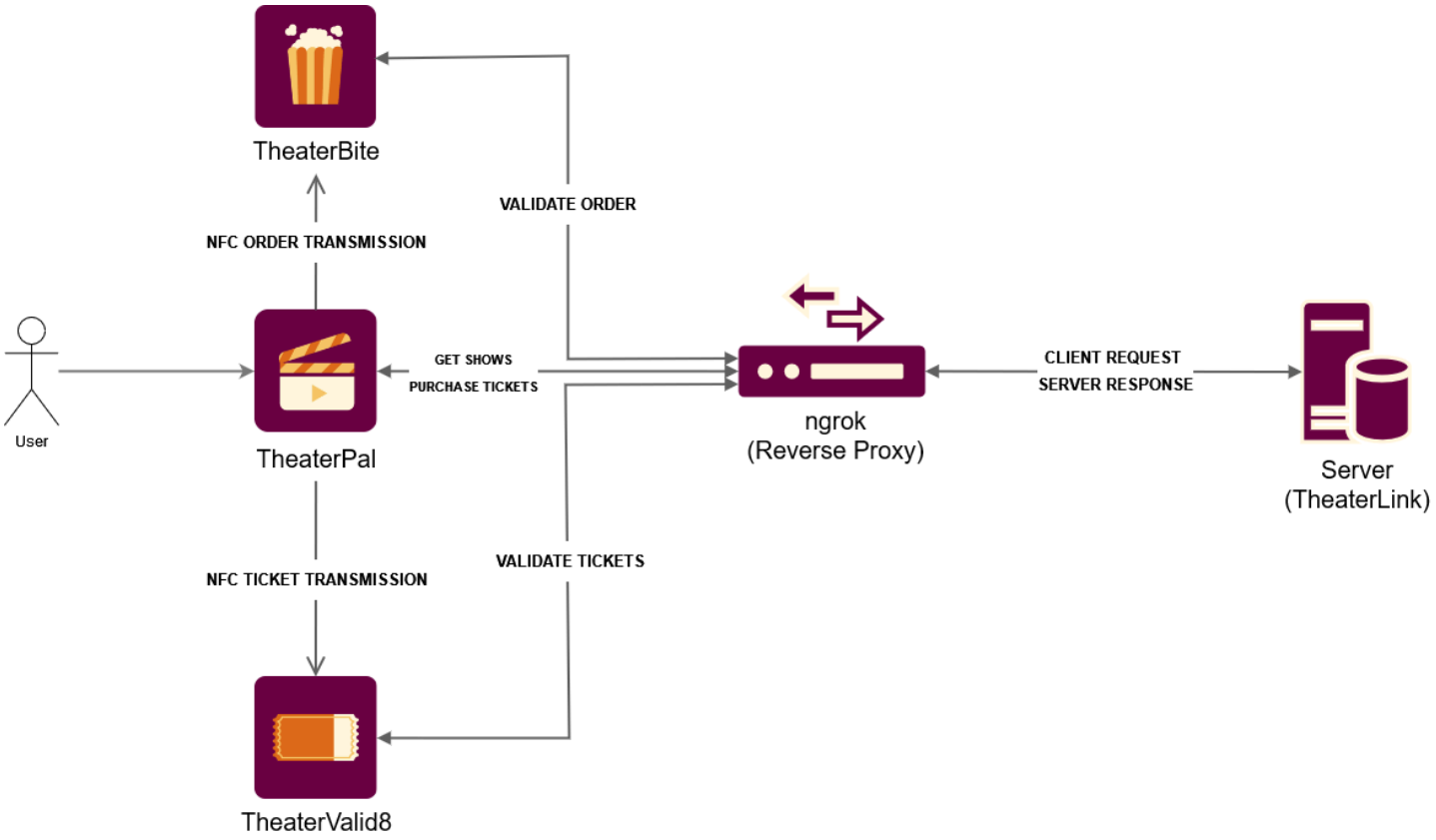
The systems is comprised by 4 main components:

Architecture of the system

Component	Description
TheaterLink	Is the backend service. Interacts with the database and provides the API for the frontend. Written in Python using Flask.
TheaterPal	The main application. Allows users to consult shows, buy tickets, buy food from the cafeteria among other features. Developed with Kotlin
TheaterValid8	The validation app for the tickets. Reads tickets using NFC from the customers and validates them. Developed with Kotlin
TheaterBite	The cafeteria terminal app that receives orders from the customers also using

Component	Description
	NFC. Developed with Kotlin

The following image describes the schema of the system:



TheaterLink / Backend Service

- To unclutter the main application, app.py, endpoints were separated into different files. Each file is responsible for a different set of endpoints.

All state changing requests are signed by the user's private key. The signature is then verified either by the server or by the other apps. This is done to ensure the integrity of the data and to prevent unauthorized access.

For example, the request `purchase_tickets` (POST), is signed by the user's private key. The server then verifies the signature using the user's public key. If the signature is valid, the request is processed. If not, the request is rejected, and an error is returned.

> This is the only POST request in which the main app directly interacts with the server and not with the other apps. All other POST requests have either the TheaterValid8 or TheaterBite apps as the intermediary.

The requests `submit_order` and `validate_tickets` are handled by the TheaterBite and TheaterValid8 apps, respectively. The signature is verified by these apps, after the user taps their phone on the terminal.

> The data sent over NFC is signed by the user's private key.

User Endpoints

user.py contains the endpoints for user management. It allows the creation of new users, login, and user information retrieval.

- API endpoints:
  - `POST /register` - Register a new user
  - `GET /get_user` - Get user information given an ID
  - `GET /users` - Get all users (unused//debugging purposes)

Show Endpoints

shows.py contains the endpoints for show management. It allows the retrieval of all shows that were previously added to the database (during creation).

Ticket Endpoints

- API endpoints:
  - `GET /shows` - Get all shows
    - The argument `images` can be passed to the query string to include the base64 encoded image of the show in the response. This is useful for displaying the image in the frontend.
    - The frontend only asks for the shows once, and then caches them.

tickets.py contains the endpoints for ticket management. It allows the creation of new tickets, retrieval of all tickets, and ticket validation.

- API endpoints:
  - `POST /purchase_ticket` - Purchase a ticket
    - Request body:

```
{
  "data:" {
    "show_date_id": "int",
    "user_id": "string",
    "num_tickets": "int",
  },
  "signature": "string"
}
```

- The response will contain the tickets in the following format:

```
{
  "tickets": [
    {
      "ticketid": "string",
      "userid": "string",
      "date": "string",
      "price": "int",
      "seat": "string",
      "showname": "string"
    }
  ]
}
```

- On top of that, a vouchers array will be returned, as per each ticket that was purchased, a voucher will be generated. The vouchers are in the following format:

```
{
  "vouchers": [
    {
      "voucherid": "string",
      "vouchertype": "string",
      "userid": "string"
    }
  ]
}
```

- Voucher type are either "free popcorn" or "free coffee", chosen randomly. A 5% voucher discount is also generated if the total price of the tickets is greater than 200.

- `GET /tickets` - Get all tickets
- `POST /validate_ticket` - Validate a ticket
  - This endpoint is called by the TheaterValid8 app to validate a ticket.

        - Request body:

```
{
  "ticketids": ["string"],
  "userid": "string"
}
```

        -
    - `POST /set_ticket_as_used` - Set a ticket as used
        - Receives a ticket ID and sets it as used. This is used when a ticket is validated by the TheaterValid8 app.

Voucher Endpoints

vouchers.py contains the endpoints for voucher management.

- API endpoints:
    - `GET /vouchers` - Get all vouchers for a user

Transaction Endpoints

transactions.py contains the endpoints for transaction management. It allows the retrieval of all transactions and other transaction-related operations.

As per the specifications, when the user consults all transctions, the server should also return the vouchers and tickets that are still not used by the user. This allows the customer to recover some voucher transmitted by mistake in a previous order, or not yet transmitted, and get rid of used ones, if they are still there.

- API endpoints:
    - `GET /transactions` - Get all transactions for a given user
        - The response will contain the transactions in the following format:

```
{
  "transactions": [
    {
      "timestamp": "timestamp",
      "transaction_id": "string",
      "transaction_type": "string",
      "total": "double",
      "vouchers_used" : ["Voucher"],
      "vouchers_generated": ["Voucher"],
      "items" : [
          // IF TYPE IS TICKET PURCHASE
          {
              "date": "string",
              "num_tickets": "int",
              "price": "double",
              "shownname": "string",
          },
          // IF TYPE IS FOOD PURCHASE
          {
              "itemname": "string",
              "price": "double",
              "quantity": "int",
          }
      ]
    }
  ],
  "tickets": [
    {
      "ticketid": "string",
      "userid": "string",
      "date": "string",
      "price": "int",
      "seat": "string",
      "showname": "string"
    }
  ],
  "vouchers": [
    {
      "voucherid": "string",
      "vouchertype": "string",
      "userid": "string",
      "isUsed": "bool"
    }
```

```
  ]
}
```

Cafeteria Endpoints

cafeteria.py contains the endpoints for cafeteria management. It allows the creation of new orders, retrieval of all orders, and other.

- API endpoints:
  - `POST submit_order`
    - Endpoint used by the TheaterBite app to submit an order when the user taps their phone on the terminal (NFC).
    - Request body:

```
{
  "vouchers_used": ["string"],
  "user_id": "string",
  "order": {
    "items": [
      {
        "itemname": "string",
        "price": "double",
        "quantity": "int"
      }
    ]
  }
  "total": "double",
}
```

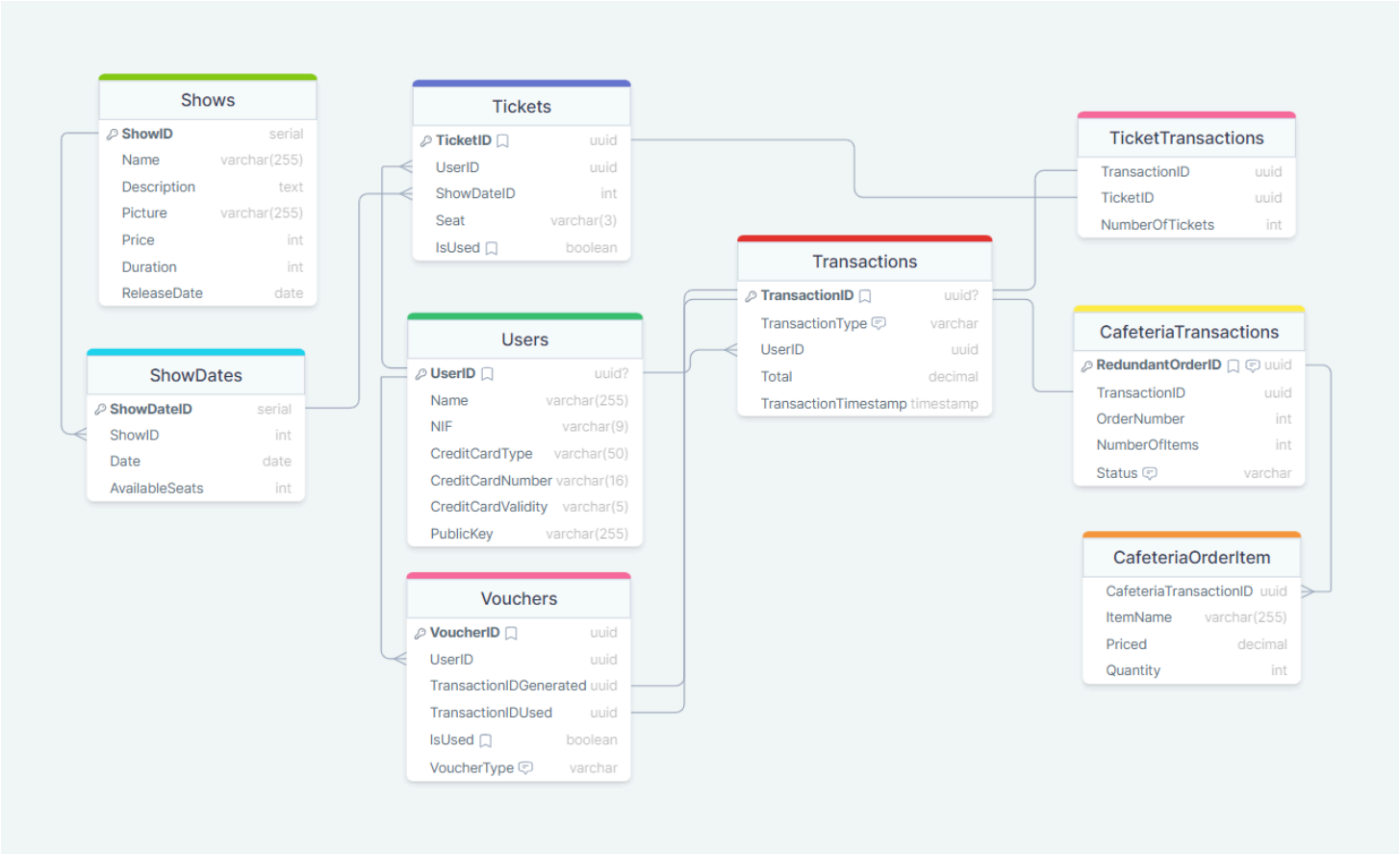
  - `GET /orders` - Get all orders

Database and Data Schemes

The database used in this project is PostgreSQL and we used the `psycopg2` library to interact with it, in the backend service.

We used ElephantSQL to host the database.

The database diagram is as follows:



For a more in-depth view of the database schema, visit this link

It is composed by 9 tables:

Users

Contains the information of the users that are registered in the system.

Column	Type	Description	Notes
`UserID`	UUID	The user's ID, generated by the system	
`Name`	VARCHAR(255)	The user's name	
`NIF`	VARCHAR(9)	The user's NIF (Fiscal Identification Number)	
`CreditCardType`	VARCHAR(50)	The user's credit card type (VISA,	

Column	Type	Description	Notes
		MasterCard, etc.)	
`CreditCardNumber`	VARCHAR(16)	The user's credit card number	
`CreditCardValidity`	VARCHAR(5)	The user's credit card expiration date (MM/YY)	
`PublicKey`	VARCHAR(255)	The user's public key	This is transmitted from the client to the server and then used for verifying signatures

Shows

Contains the information of the shows that are available for purchase.

Column	Type	Description	Notes
`Showid`	SERIAL	The show's ID, incremented by the system	
`Name`	VARCHAR(255)	The show's name	
`Description`	TEXT	The show's description	
`Picture`	VARCHAR(255)	The show's picture internal URL	
`Price`	INT	The price of the tickets for the show	
`Duration`	INT	The duration of the show in minutes	
`ReleaseDate`	DATE	The date the show was released	

ShowDates

Contains the dates that a show is available.

Column	Type	Description	Notes
`ShowDateID`	SERIAL	The show date's ID, incremented by the system	

Column	Type	Description	Notes
`ShowID`	INT	The show's ID	Foreign key to the `Shows` table
`Date`	DATE	The date the show is available	
`AvailableSeats`	INT	The number of available seats for the show	Not used in the current implementation

Transactions

Contains the information of the transactions that were made.

The type of the transaction is an ENUM with the following values: `TICKET_PURCHASE`, `CAFETERIA_ORDER`.

Column	Type	Description	Notes
`TransactionID`	UUID	The transaction's ID, generated by the system	
`UserID`	UUID	The user's ID	Foreign key to the `Users` table
`TransactionType`	transaction_type	The type of the transaction	
`Total`	DOUBLE	The total amount of the transaction	
`TransactionTimestamp`	TIMESTAMP	The timestamp of the transaction	

Tickets

Contains the information of the tickets that were purchased.

Column	Type	Description	Notes
`TicketID`	UUID	The ticket's ID, generated by the system	
`UserID`	UUID	The user's ID	Foreign key to the `Users` table

Column	Type	Description	Notes
`ShowDateID`	INT	The show date's ID	Foreign key to the `ShowDates` table
`Seat`	VARCHAR(5)	The seat allocated to the ticket	
`isUsed`	BOOLEAN	Whether the ticket was used or not	Default is `FALSE`

Vouchers

Contains the information of the vouchers that were generated.

The type of the voucher is an ENUM with the following values: `FREE_POPCORN`, `FREE_COFFEE`, `FIVE_PERCENT`.

Column	Type	Description	Notes
`VoucherID`	UUID	The voucher's ID, generated by the system	
`UserID`	UUID	The user's ID	Foreign key to the `Users` table
`VoucherType`	voucher_type	The type of the voucher	
`isUsed`	BOOLEAN	Whether the voucher was used or not	Default is `FALSE`
`TransactionIDGenerated`	UUID	The transaction that generated the voucher	Foreign key to the `Transactions` table
`TransactionIDUsed`	UUID	The transaction in which the voucher was used	Foreign key to the `Transactions` table

TicketTransactions

Contains the information of the tickets that were purchased in a transaction.

Column	Type	Description	Notes
`TransactionID`	UUID	The transaction's ID	Foreign key to the `Transactions` table
`TicketID`	UUID	The ticket's ID	Foreign key to the `Tickets` table
`NumberOfTickets`	INT	The number of tickets purchased	

Note: If more than one ticket was purchased in a transaction, the ticket ID will be set to the first ticket purchased and the number of tickets will be the total number of tickets purchased.

CafeteriaTransactions

Contains the information of the order that was made in the cafeteria.

The status of an order is an ENUM with the following values: `COLLECTED`, `PREPARING`, `READY`, `DELIVERED`.

Column	Type	Description	Notes
`TransactionID`	UUID	The transaction's ID	Foreign key to the `Transactions` table
`RedundantOrderID`	UUID	A redundant order ID	
`OrderNumber`	INT	The order number	Will be used to identify the order in the cafeteria
`NumberOfItems`	INT	The number of items in the order	
`Status`	order_status	The status of the order	

> Note: The `RedundantOrderID` is needed because `CafeteriaOrderItem` needs to reference the `CafeteriaTransactions` table, and the `TransactionID` is not the primary key there.
> This could have been avoided by using the `TransactionID` as the foreign key in the `CafeteriaOrderItem` table, but we decided to keep the consistency of the database, even if it meant adding redundancy.

CafeteriaOrderItem

Contains the information of the items that were purcased in an order.

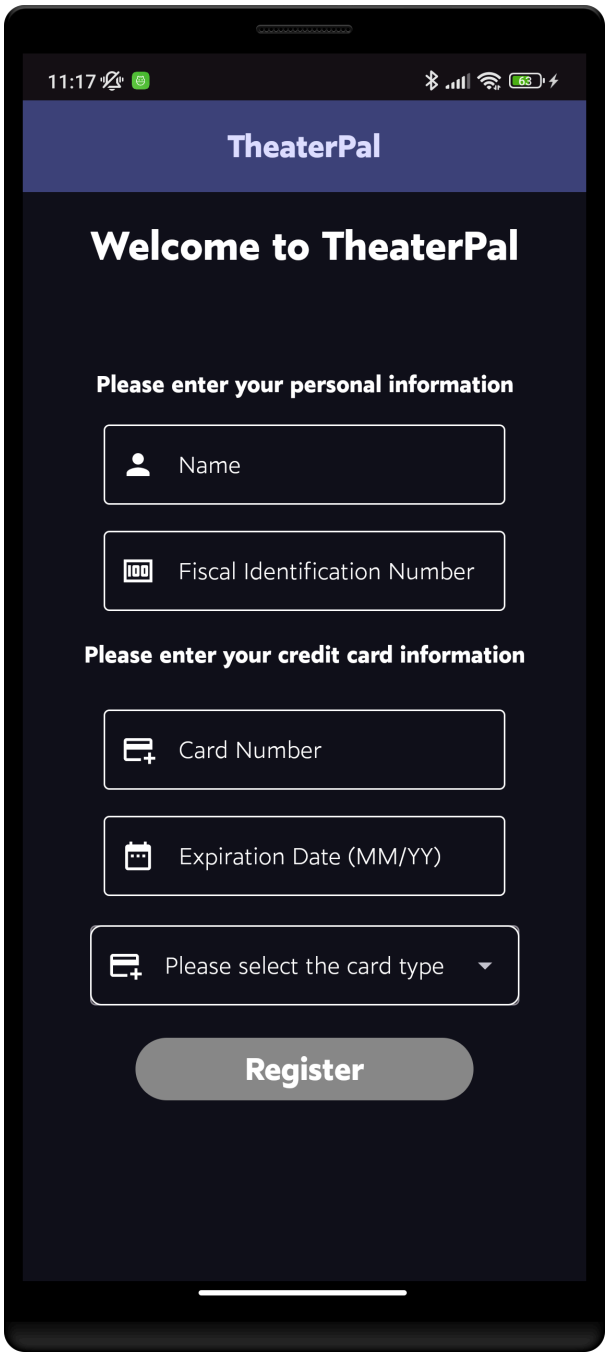
Column	Type	Description	Notes
`CafeteriaTransactionID`	UUID	The cafeteria transaction's ID	Foreign key to the `CafeteriaTransactions` table
`ItemName`	VARCHAR(255)	The name of the item	
`Price`	DOUBLE	The price of the item	
`Quantity`	INT	The quantity of the item	

Features / How To Use

Register

The first time a customer uses the system, they must register. This is done by providing their name, NIF, and credit card information. The credit card information is used to make the payments for the tickets and cafeteria orders.

The image below shows the registration screen:



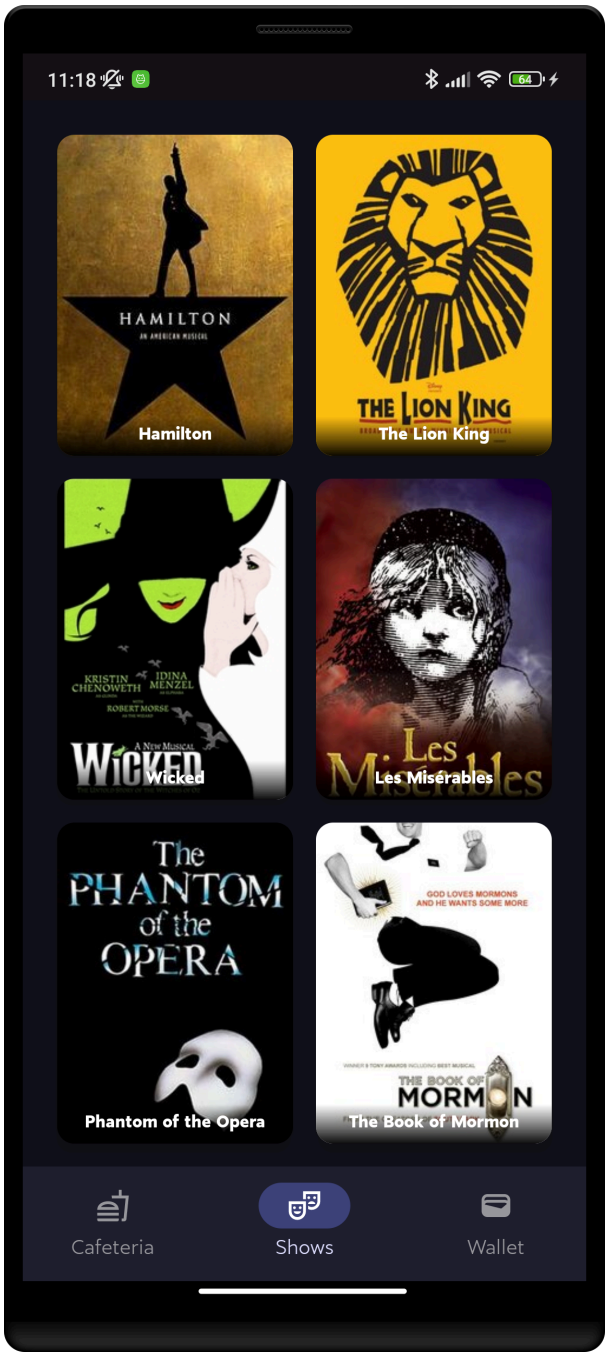
After the user registers, an RSA key pair is generated. The public key is sent to the server and stored in the database. The private key is stored in the user's device.

This step is only done once, and the user can then use the system without having to register again.

Consult Shows

The customer, after registering, is presented with a list of shows that are available for purchase. The shows are retrieved from the server and displayed in the app.

The image below shows the list of shows:



Purchase Tickets

By clicking on a show card, the user is presented with more information about the show, such as the description, price, and duration:

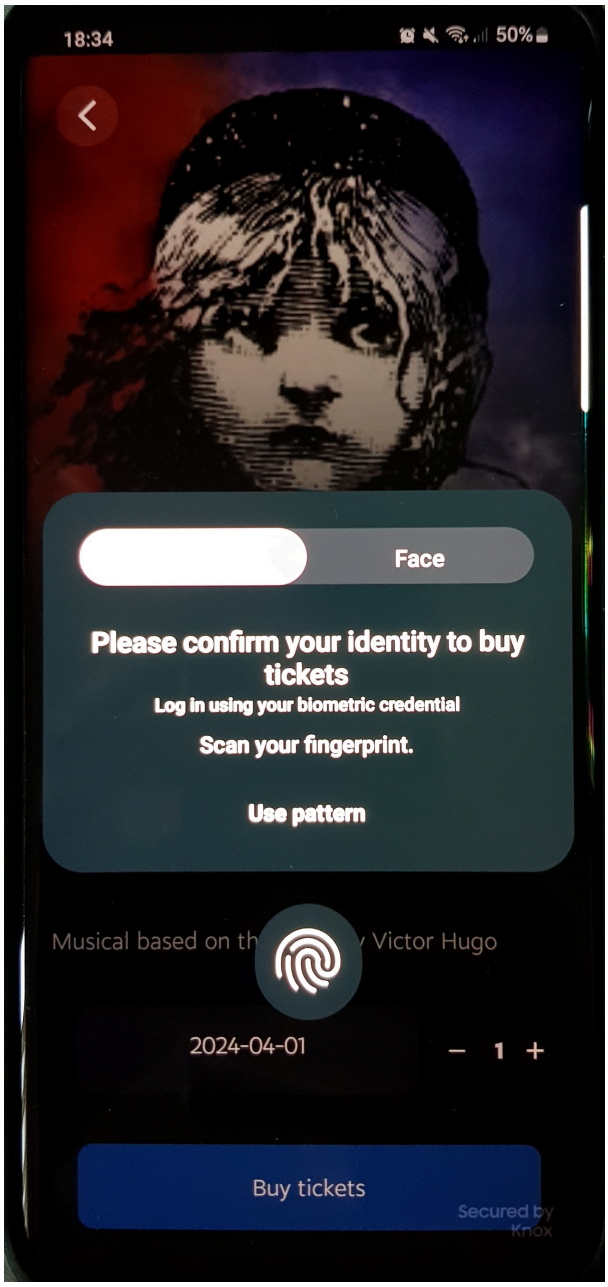
From here, the user can select from a list of available dates and purchase tickets for the show (maximum of 4 tickets per purchase).

Biometric Authentication

Before submitting the order, the user must authenticate using biometrics. This is done to ensure that the user is the one making the purchase.

> Note: As some phones don't provide either a fingerprint sensor or facial recognition, we also provide the option to authenticate using a PIN or pattern.

The authentication screen is shown below:

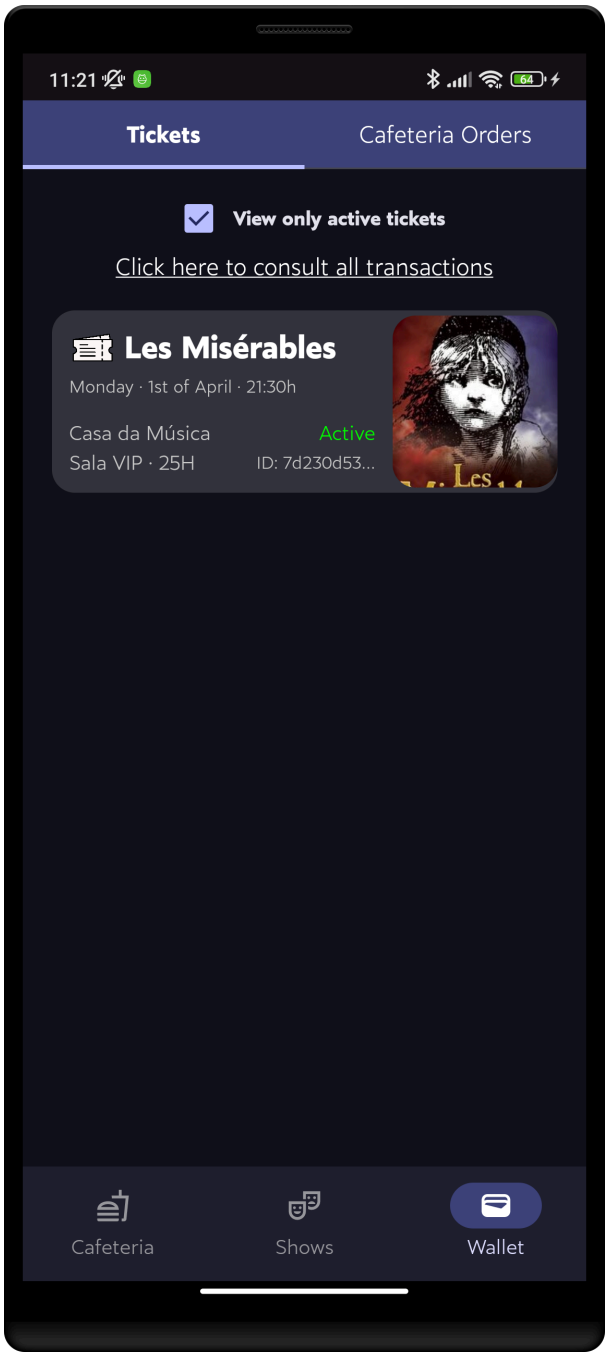


After the user authenticates, the order is submitted to the server, and the tickets are generated.

Consult Tickets

The user can consult all the tickets they have purchased. This is done by clicking on the "Wallet" icon in the bottom navigation bar.

The image below shows the wallet screen:



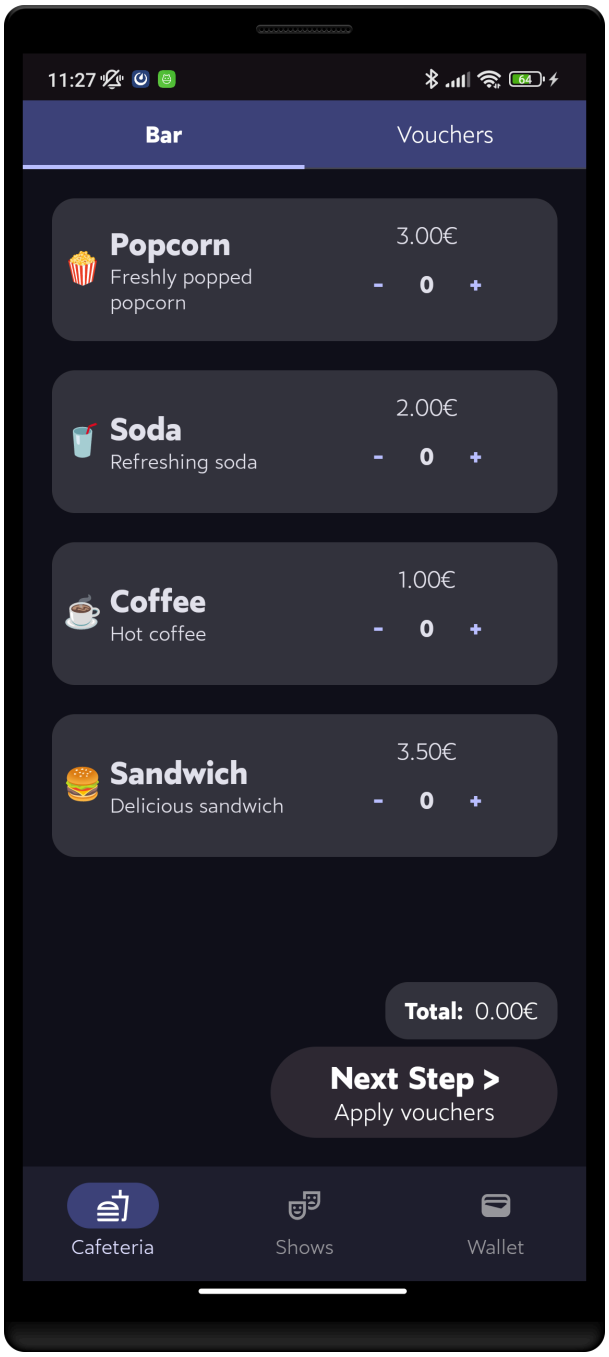
The user can also filter the tickets to show only the ones that are still valid.

The wallet screen has two tabs: one for the tickets and other for orders made in the cafeteria.

Check items in the Cafeteria

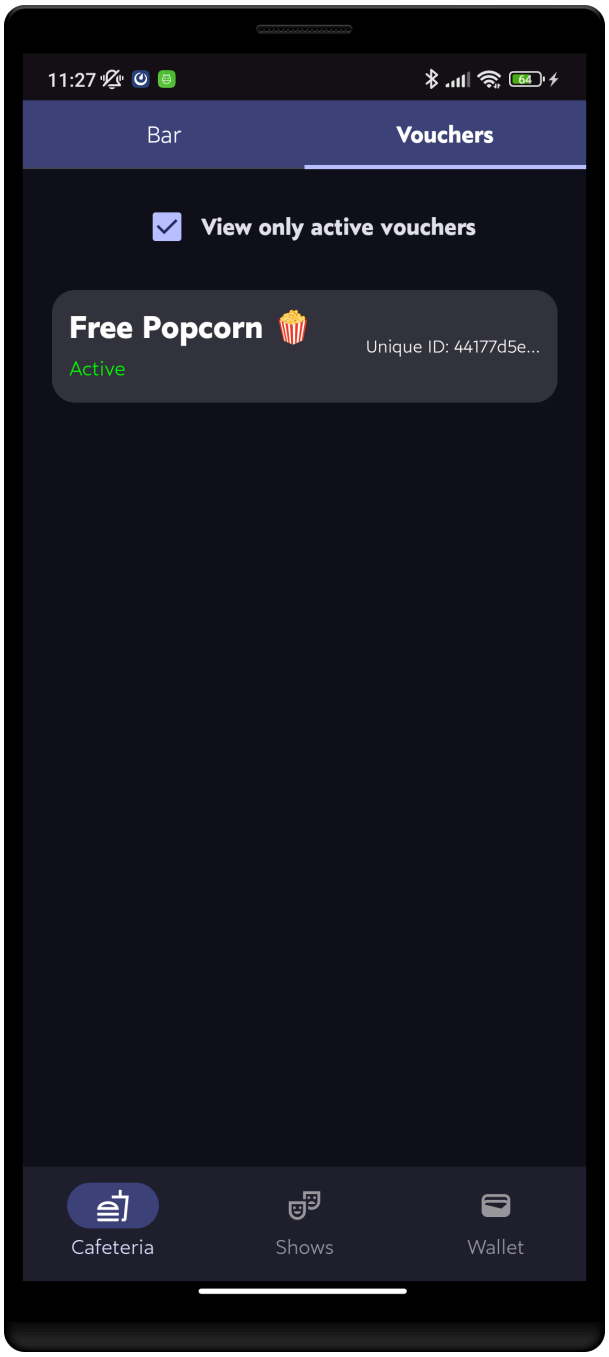
The user can also check the items available in the cafeteria. This is done by clicking on the "Cafeteria" icon in the bottom navigation bar.

The image below shows the cafeteria screen:



Consult Available Vouchers

By swiping to the "Vouchers" tab in the cafeteria screen, the user can consult all the vouchers they have available:

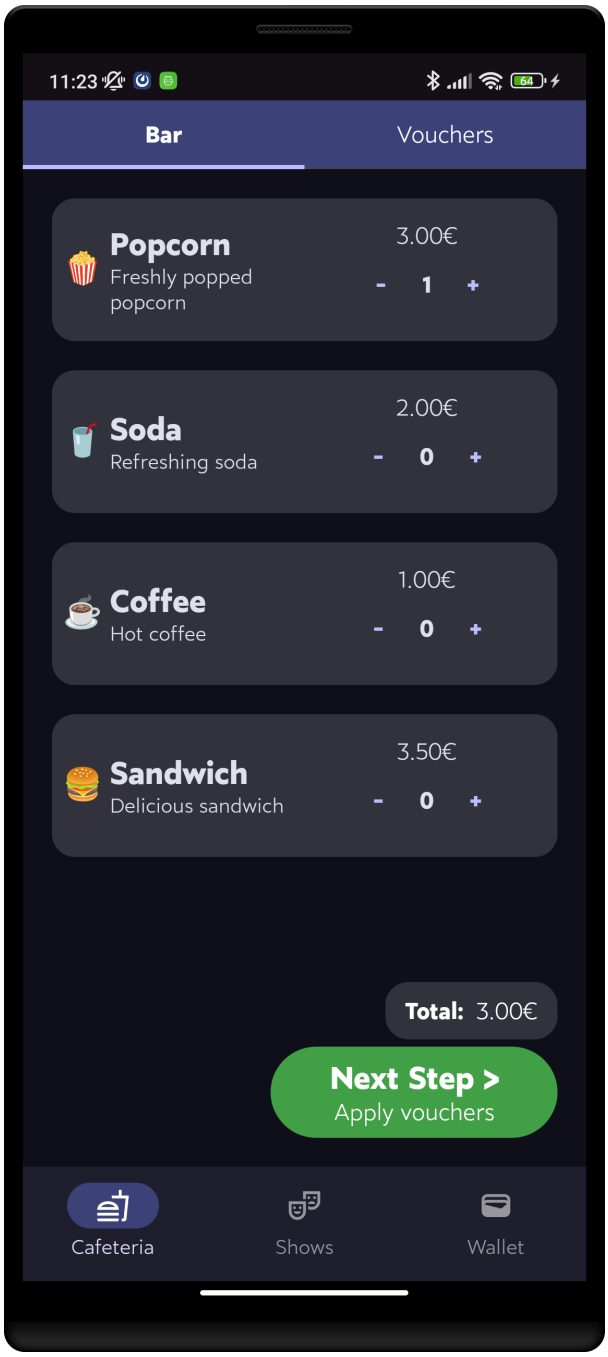


The user can also filter the vouchers to show only the ones that are still valid.

Order Food

To make an order in the cafeteria, the user first selects from the list of products:

Choosing Products



Note that the button to submit the order is disabled until the user selects at least one item.
At any moment, the user can see the total price of the order.

Select Vouchers

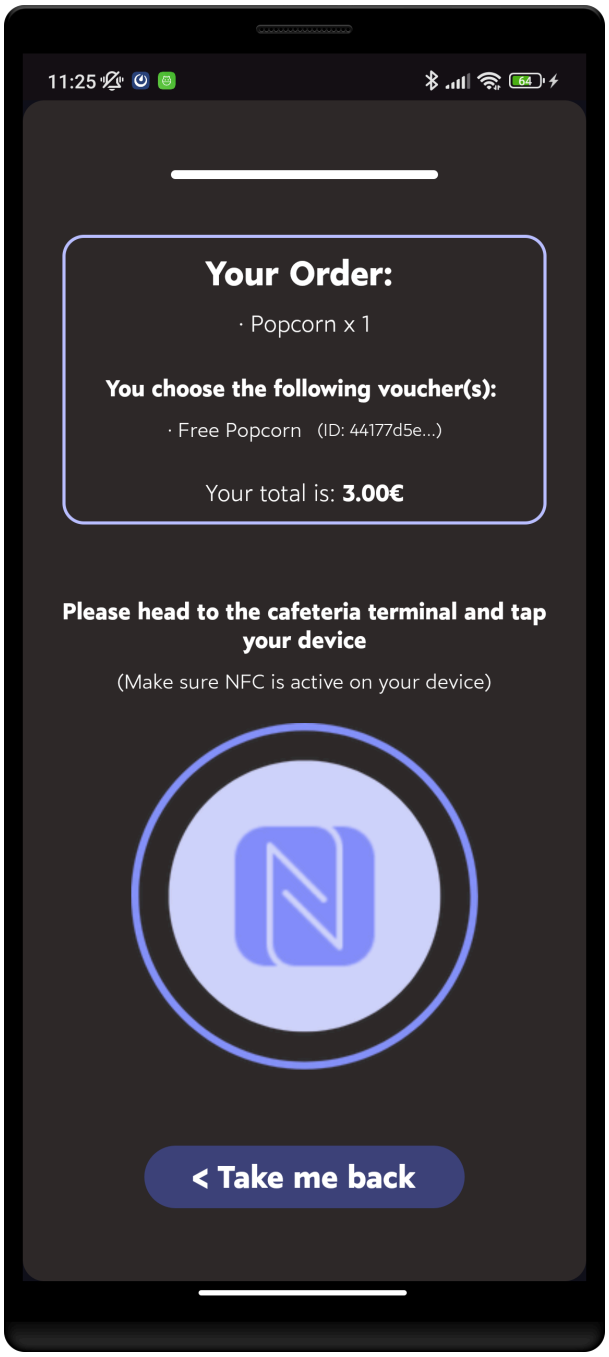
After clicking on the "Next Step" button, the user is presented with a screen to select the vouchers they want to use:

There are some restriction in place:

- The user can only select up to 2 vouchers.
- If available, the user can select only one 5% discount voucher.

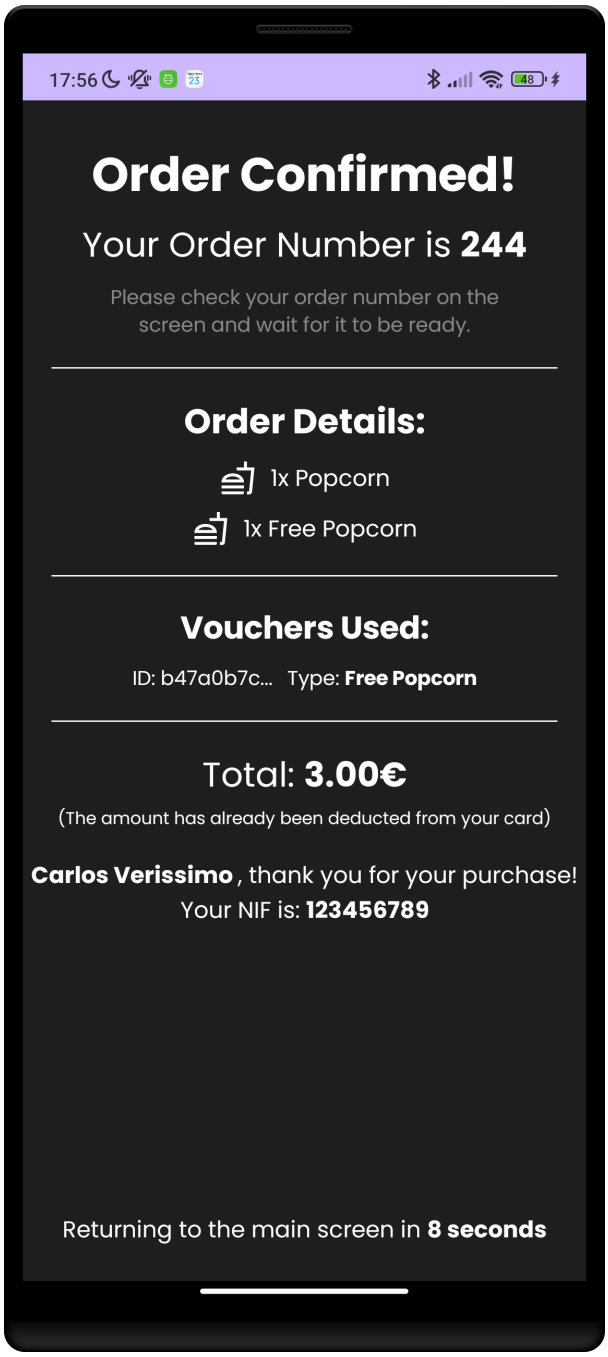
Validate Order

After selecting the vouchers, the user can click on the "Submit Order" button to generate the order and, go to the cafeteria validation screen.



At this point, NFC should be enabled. If it isn't, the user is prompted to enable it before proceeding.

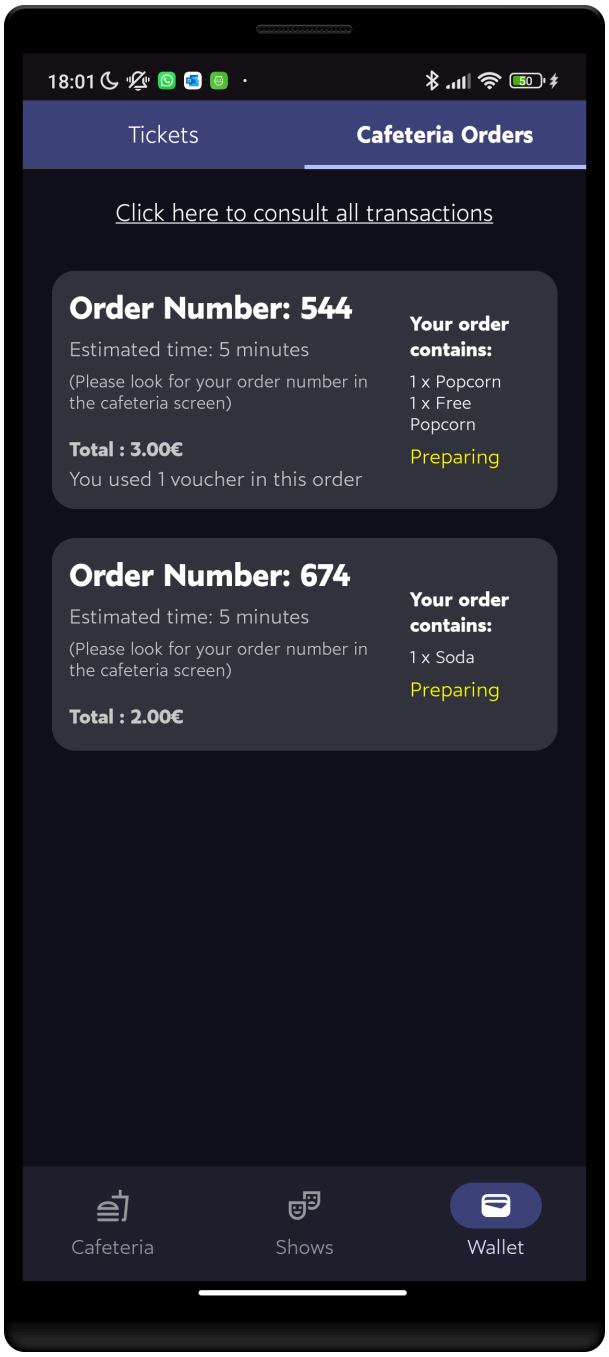
After the validation is done, the user will see in the cafeteria terminal a screen that shows the result of the validation. It will show the order number that the customer should look for in the cafeteria.



The terminal will show the items in the order and the total price, as well as the vouchers that were used.

Consult Orders

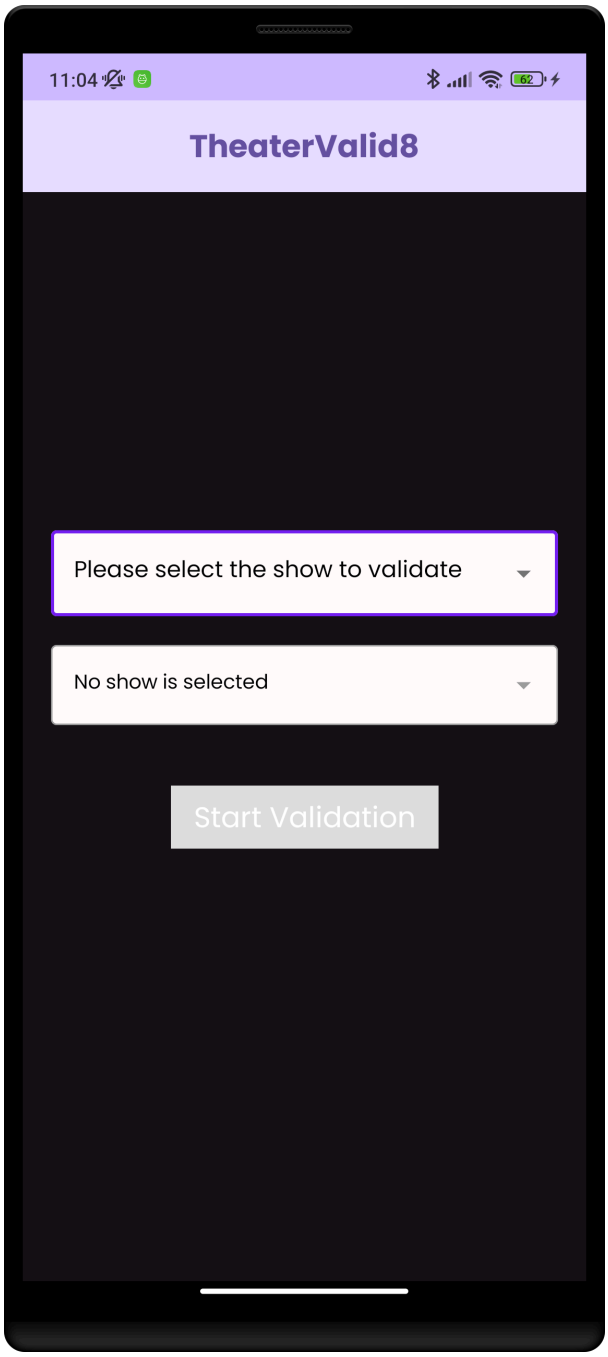
After the order is submitted, the user can consult all the orders they have made in the cafeteria. This is done by clicking on the "Wallet" icon in the bottom navigation bar and swiping to the "Orders" tab.



Validate Tickets

Prepare Terminal (Server / Terminal)

In order to validate tickets, the terminal app should be set to accept tickets for a given show at a given date. The following screen is the first thing that the validator is presented with when opening the `TheaterValid8` app.



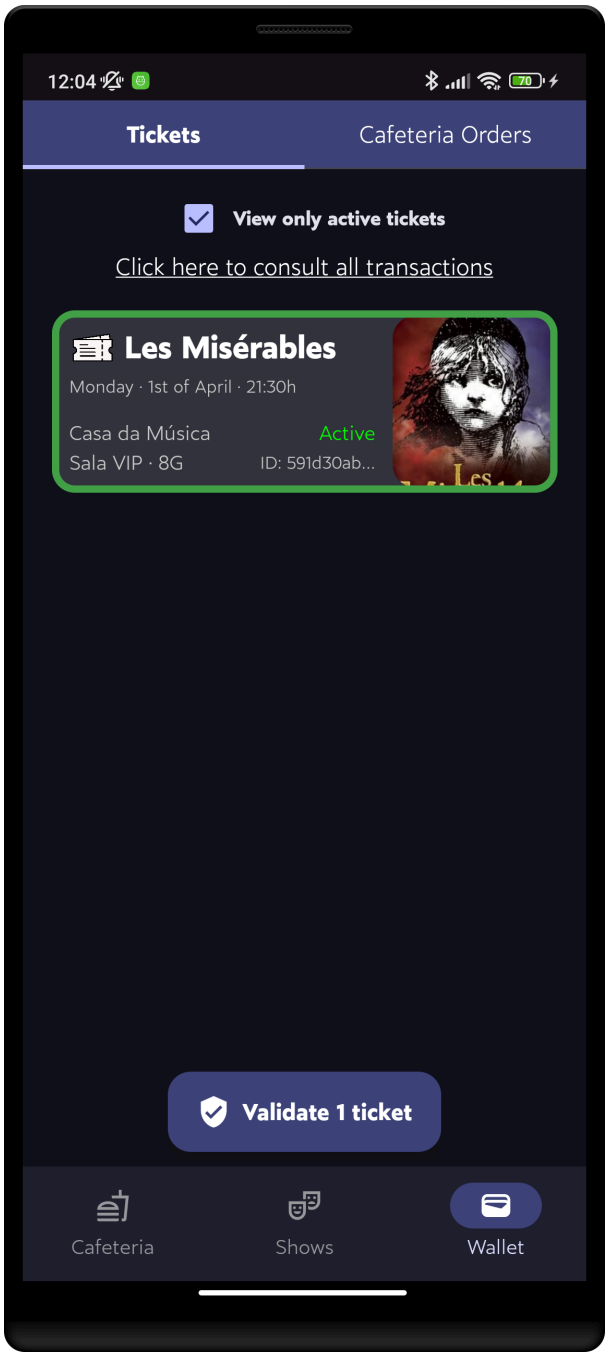
After selecting the show and date, the validator confirms the selection:

And then the following screen is presented in the terminal, as has some information about the show.

To validate tickets, just click on the "Scan" button and the terminal is ready to receive the tickets.

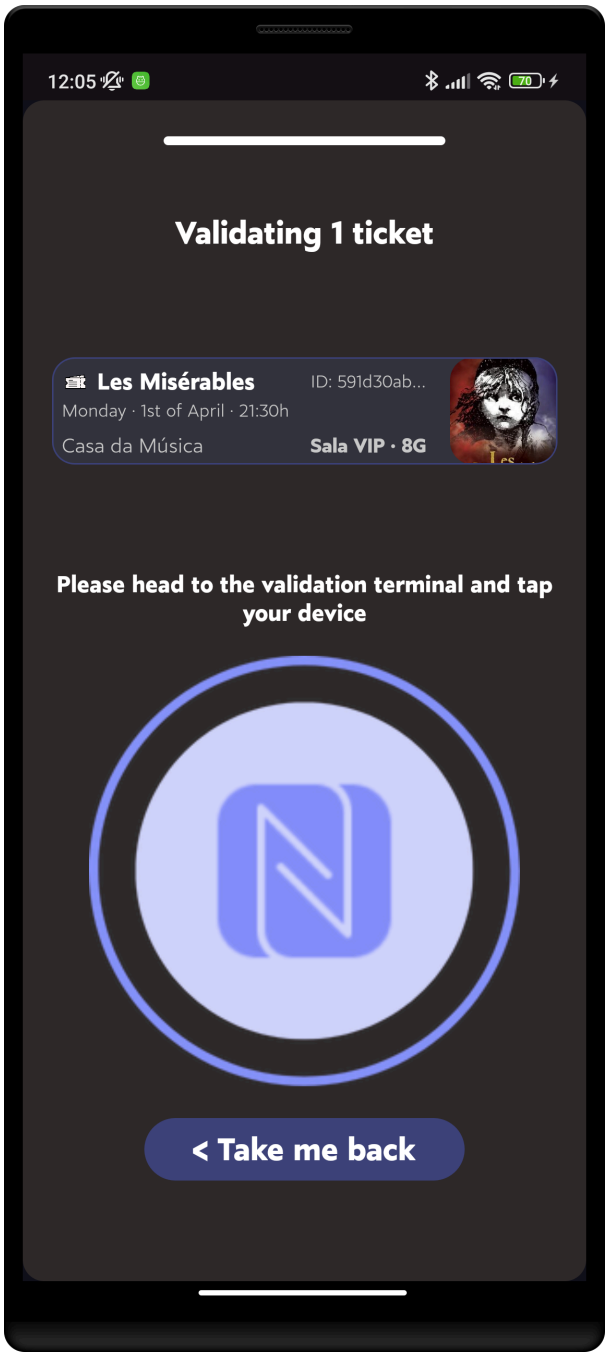
Present Tickets (Client / Customer)

The user can also validate the tickets they have purchased. In the "Wallet" screen, the user can swipe to the "Tickets" tab and, by selecting a ticket, they can validate it.



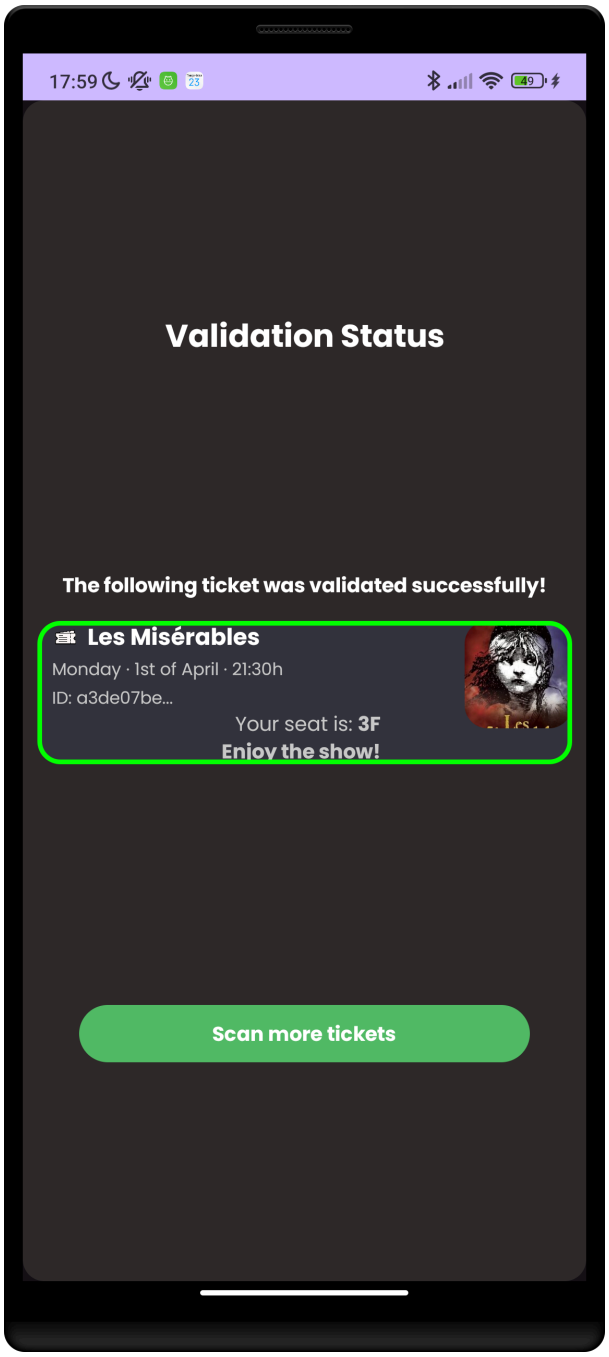
When the user selects a ticket, a button to validate it appears. At this stage, the user can chosoe up to 4 tickets to validate, at each time.

After clicking on the "Validate" button, the user is presented with a screen to tap their phone on the NFC terminal.



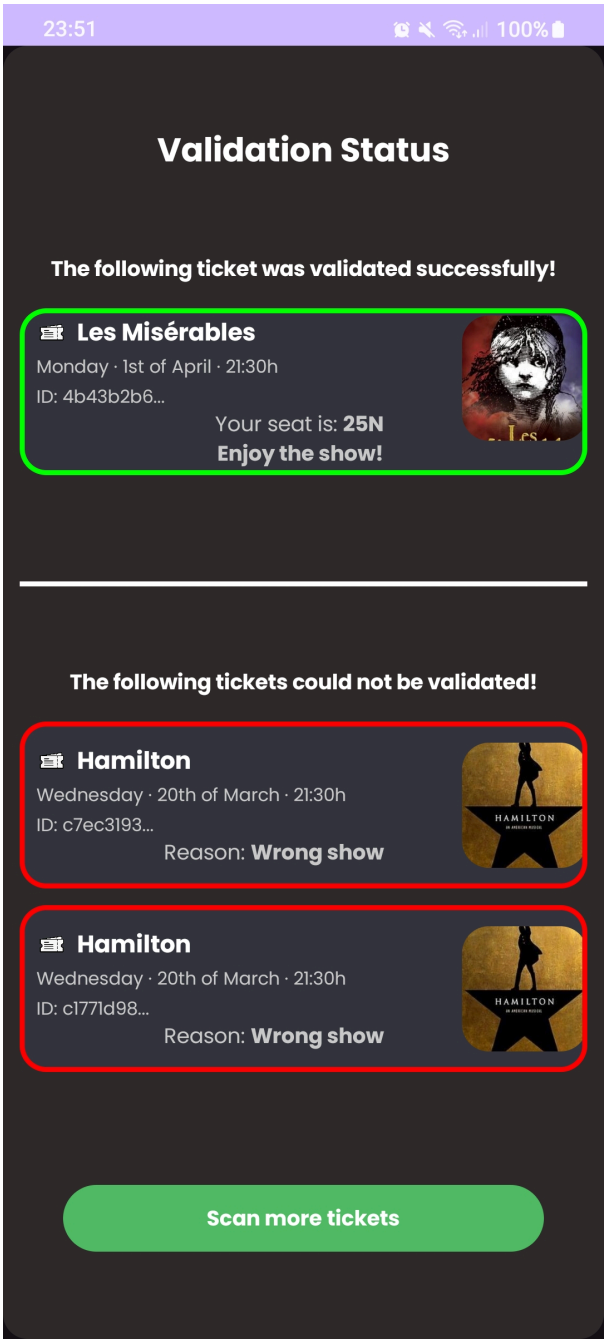
Validation Result

After receving a tap from the customer, the terminal will show the result of the validation.



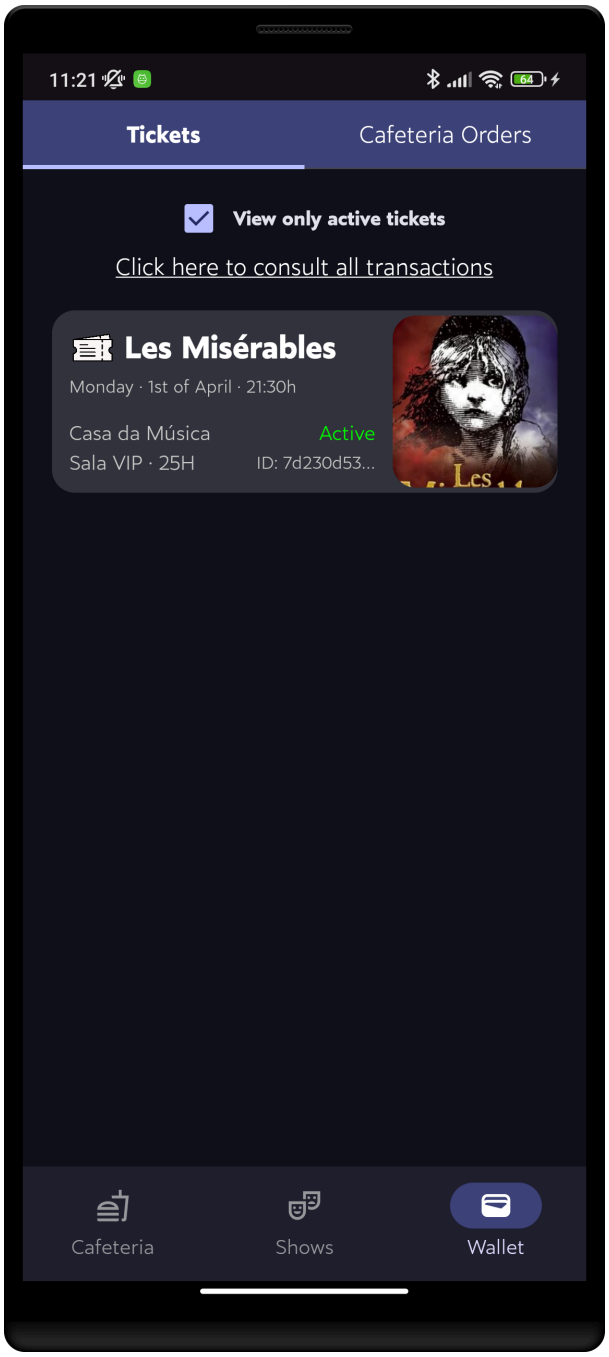
Keep in mind that the user can send up to 4 tickets at the same time, including different shows and dates.

The validation terminal is only set to accept tickets for a certain show and date. If the ticket is not valid, the user is presented with an error message.



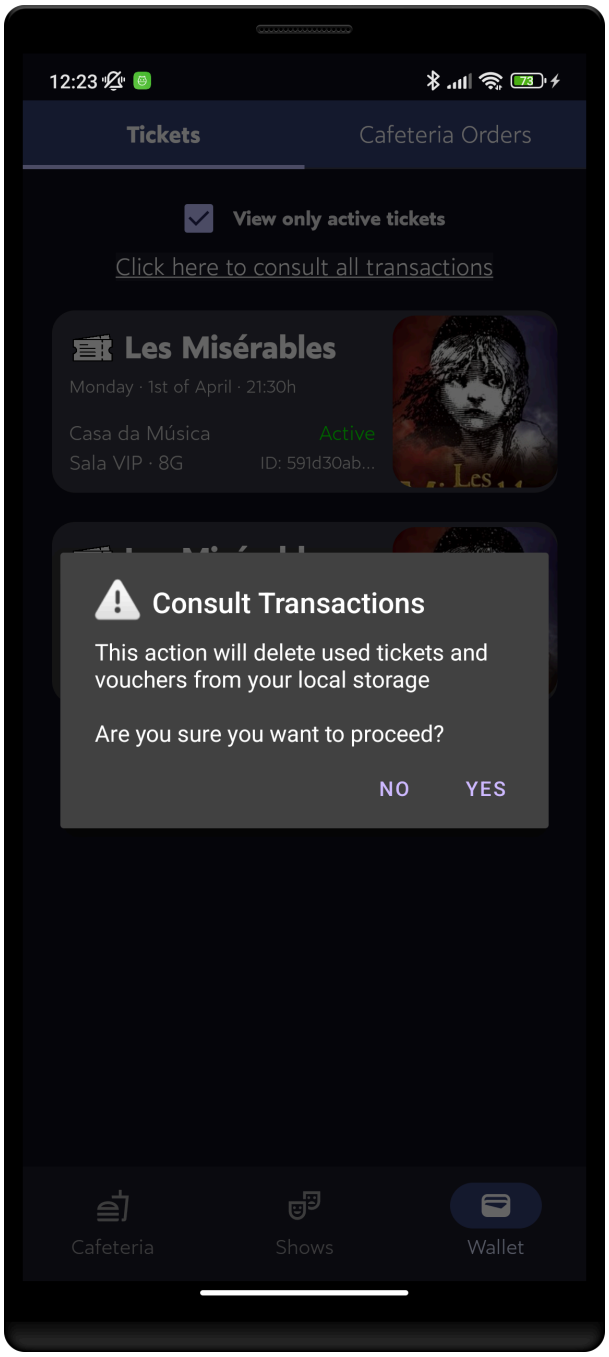
Consult Transactions

At any given moment, the user can consult all the transactions they have made. To do this, go to the "Wallet" screen and click on the `Click here to see consult transactions` text.

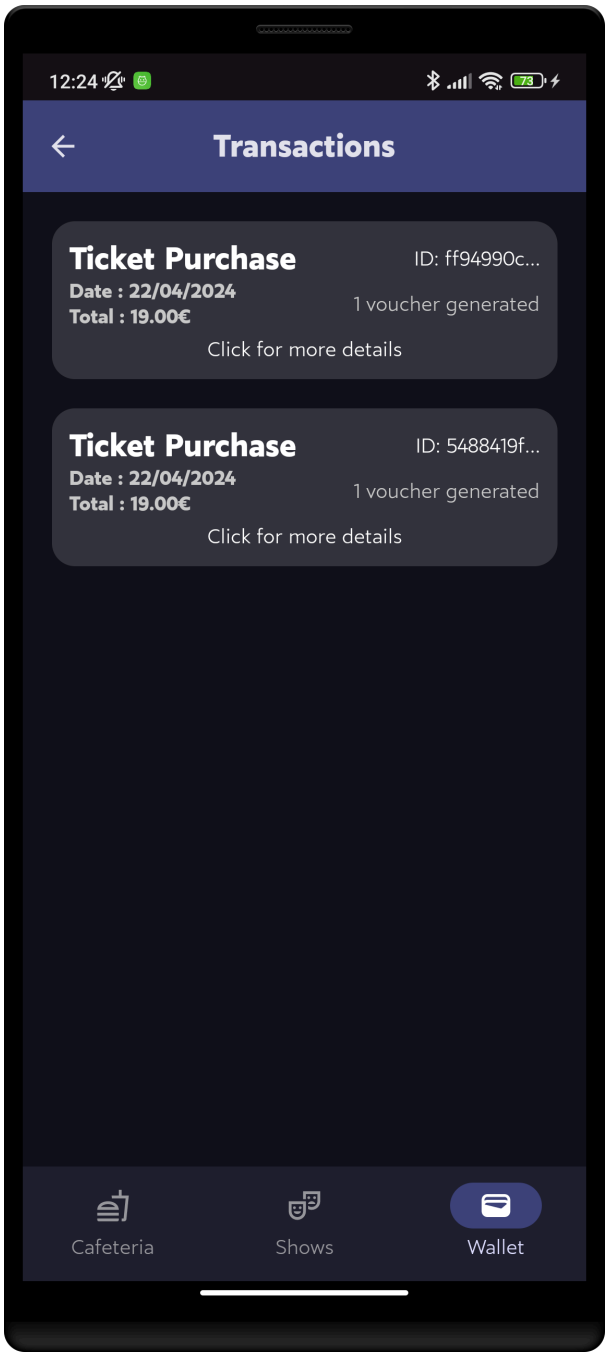


Per the specifications, consulting the transctions will fetch the vouchers and tickets that are still not used by the customer. In this way used tickets and vouchers are deleted from the customer app, allowing the customer to recover some voucher transmitted by mistake in a previous order, or not yet transmitted, and get rid of used ones, if they are still there.

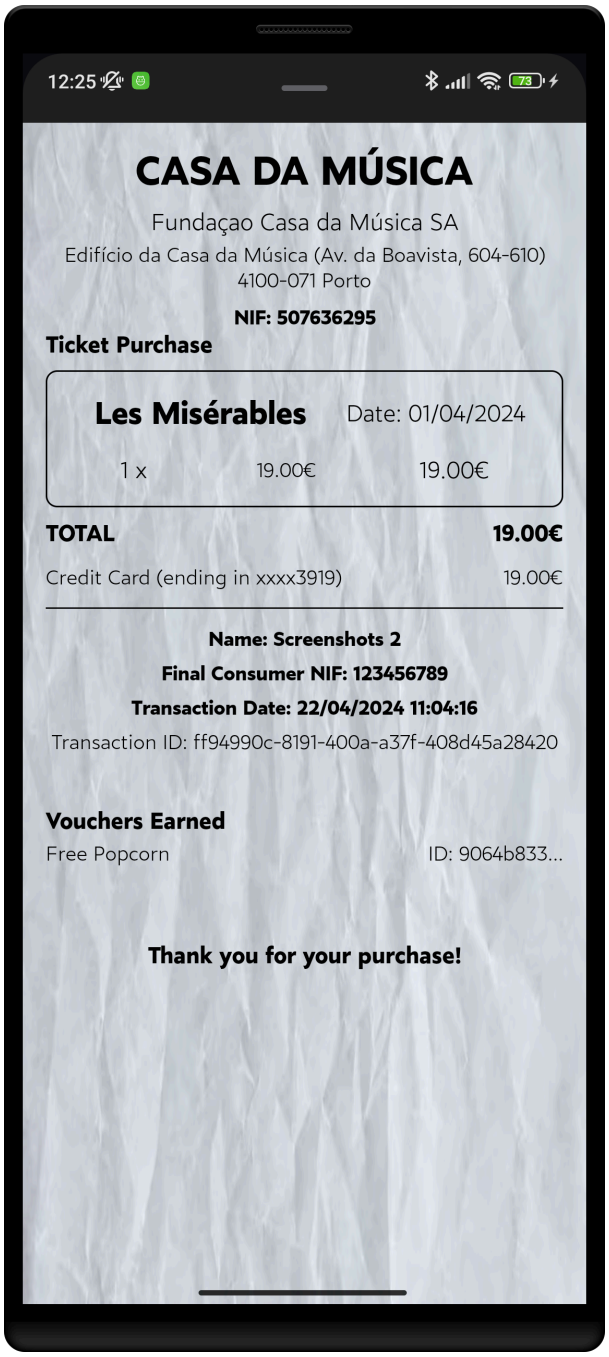
As this is a potentially heavy operation, the user is prompted to confirm if they want to fetch the transactions.



If the user chooses to proceed, the transactions are fetched and displayed in a list:



Clicking on a transaction will show a receipt-like screen with all the details of the transaction:



Scenario Tests

An internet connection is only needed for three main operations:

1. Registering for the first time
2. Purchasing tickets
3. Consult all transactions

There are some operations that can be done offline, and only require an NFC equipped phone:

Offline Ticket Validation

One of the main features of the system is the ability to validate tickets using NFC, even if the phone has no internet connection.

The scenario test is as follows:

1. The user buys tickets for a show. (With internet connection)
2. The server response contains the tickets and vouchers. The app caches the tickets and vouchers.
3. The user loses internet connection but still has the tickets and vouchers cached.
4. The user goes to the theater and tries to validate the tickets using NFC.
5. The validation is successful, and the user is allowed to enter the theater, even without internet connection

> the validation terminal needs internet connection to verify the tickets with the server.

Offline Cafeteria Ordering

Another main feature of the system is the ability to order food in the cafeteria using NFC, even if the phone has no internet connection.

The scenario test is as follows:

1. The user chooses the items they want to order in the cafeteria.
2. They can add up to 2 vouchers to the order, even without internet connection, as they were cached when the user purhcased the tickets.
3. The user submits the order using NFC.
4. The order is successfully submitted, and the user is given an order number to look for in the cafeteria.

> The cafeteria terminal needs internet connection to submit the order to the server.

User Authentication

Another scenario test is the user authentication. The user needs to authenticate using biometrics before purchasing tickets, as this is the only way to ensure that the user is the one making the purchase.

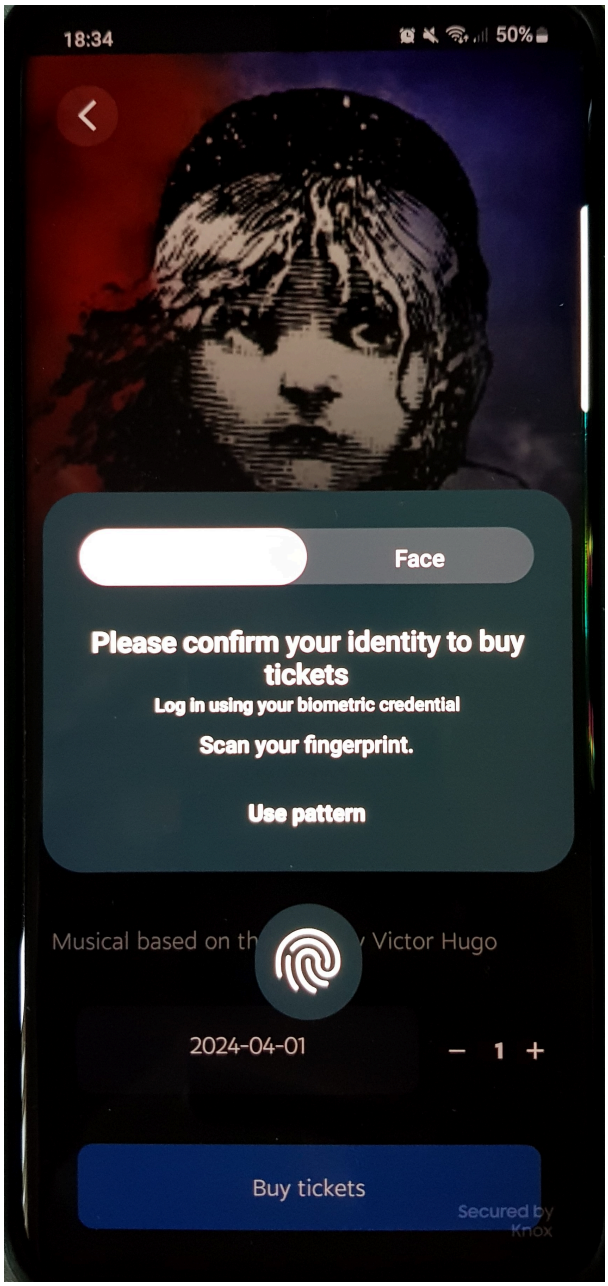
> To make a cafeteria order, this is not needed as the user needs to go up to the terminal to submit the order.

In order to support as many devices as possible, we also provide the option to authenticate using a PIN or pattern.

Thus, the authentication types are a Class 2 biometric `BIOMETRIC_WEAK`, that allows for both facial recognition and fingerprint, and a `DEVICE_CREDENTIAL` type that allows for PIN, pattern or password.

> If the user has not defined an authentication method, the app will prompt the user to do so and won't allow the user to proceed without it.

The authentication screen is shown below:

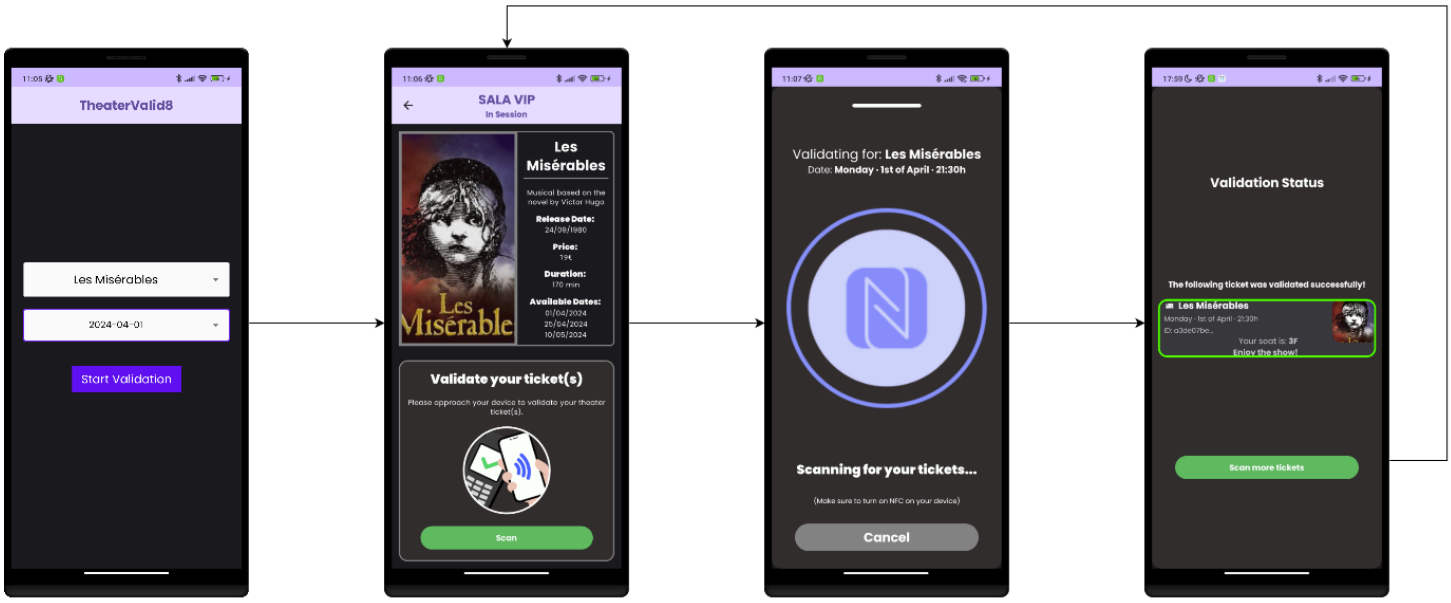


More info about biomtric authentication can be found here

Navigation Maps

TheaterPal

TheaterValid8



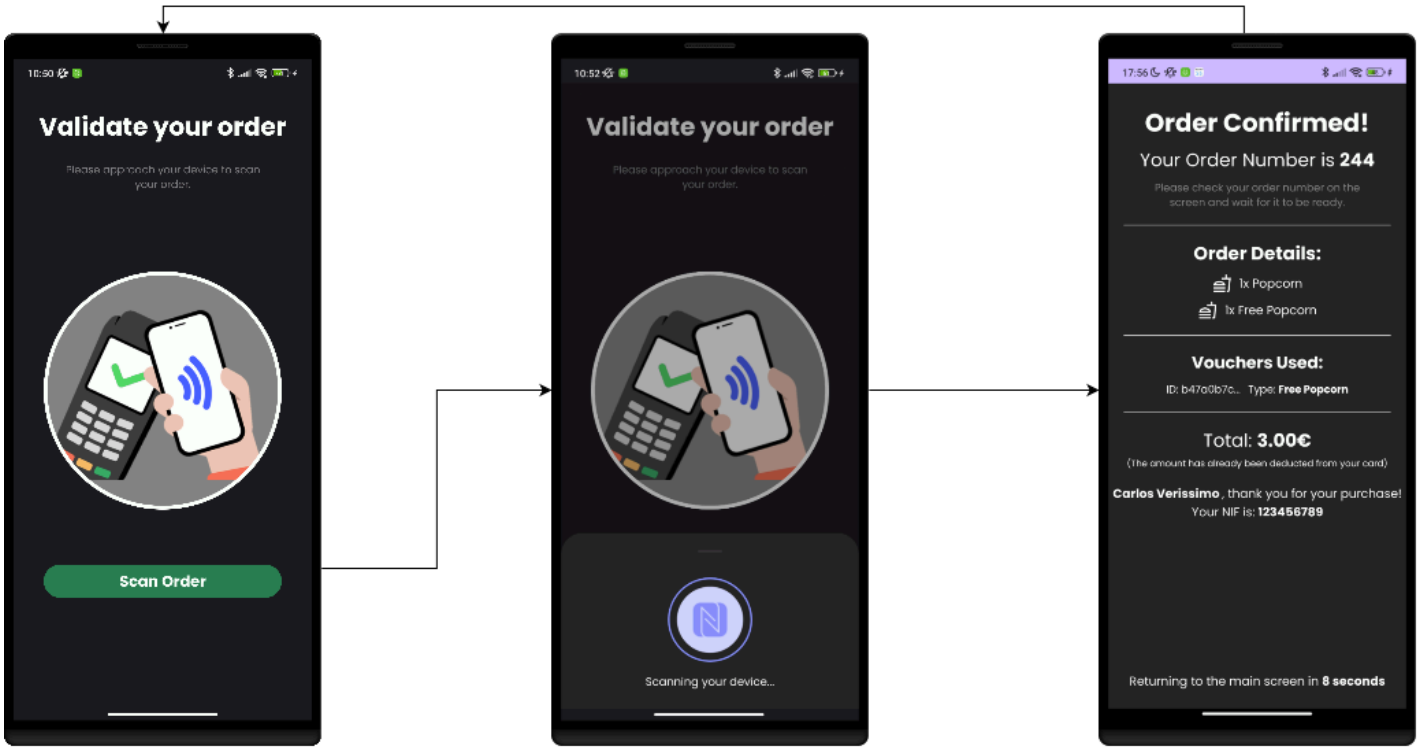
Main Activity

Validator Activity

NFC Scanning Activity

NFC Scanning Results Activity (Fragment?)

TheaterBite



Main Activity

NFC Scanning Fragment

NFC Scanning Result Activity (Fragment?)