U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MC-MOBILE COMPUTING

Report for Mobile Computing Project

Ticket / Cafeteria Ordering System

Developers:

Ricardo Ribeiro (up202310095)

Ruben Viana (up202005108)

Chaimaa Mounir (up202311347)

Summary

Introduction

The system is designed to enhance customer experience by streamlining the process of ticket purchasing, entry validation, and cafeteria service through an integrated Android-based application.

Our goal was to create a user-friendly, secure, and efficient digital solution that allows customers to manage their visits to the theatre from their mobile devices, ensuring convenience and improving service delivery.

Objective

The system comprises multiple components including a customer-facing mobile app, ticket validation terminal, and cafeteria order terminal, all interconnected via a server backend. This setup is intended to:

- Facilitate the easy purchase and validation of tickets.
- Enable the ordering and redemption of food and beverages.
- Enhance customer engagement through digital vouchers and offers.
- Ensure a good level of data security and user privacy.

Architecture

Components

TICKETO - Customer App

This application utilises the Model-View-ViewModel (MVVM) architecture to enhance maintainability and scalability. It allows customers to register, view upcoming shows, purchase tickets, redeem vouchers, place cafeteria orders and see older purchases made in the app. Built with Jetpack Compose, the app offers a reactive user interface that enhances user experience. It stores personal data and handles transactions with cryptographic keys for data integrity.

Cafeteria Order Terminal

This terminal processes orders made through the customer app, validates vouchers, and displays order summaries including the final price after voucher redemption. Utilising Jetpack Compose helps create an intuitive user interface for the cafeteria staff and clients.

Ticket Validator Terminal

This terminal scans tickets from the customer app and communicates with the server to validate them. The interface for this terminal is also developed using Jetpack Compose, ensuring a consistent look and feel across different components of the system.

REST Service

The server-side component is built using Flask, a Python web framework that facilitates the development of RESTful APIs. Hosted on the theatre's server (PS supposition), it handles customer registration and validation, ticket validation and

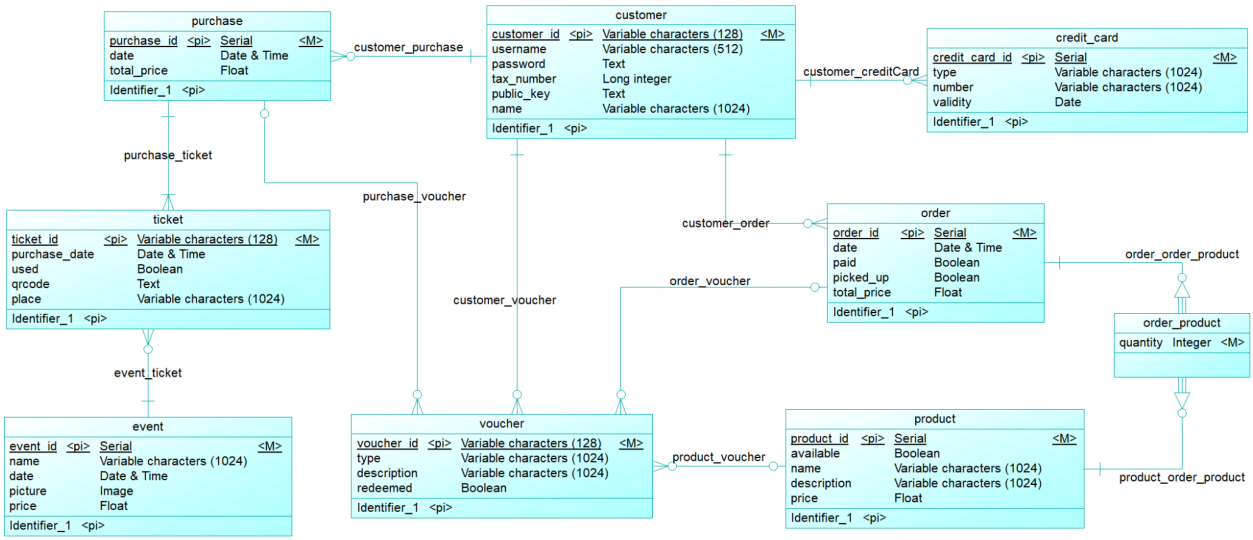
sales, voucher management, and transaction logs, ensuring scalable server-client communication.

Technologies used

- **Jetpack Compose:** Utilized for developing both the customer app and the terminals, Jetpack Compose (Google for Developers, n.d.) is a toolkit for building native Android UIs. It simplifies UI development with a declarative approach, reducing boilerplate code and enhancing app performance.
- **MVVM Architecture:** The customer app leverages the Model-View-ViewModel (MVVM) architecture, which facilitates a clean separation of concerns and data handling. This architecture enhances the scalability and maintainability of the app, allowing for efficient management of UI components and data bindings, which is necessary for real-time updates and interactive features.
- **Flask:** Flask (Pallets, n.d.) is a Python micro-framework that simplifies the creation of web services. Flask supports the system's need to manage complex interactions like user registrations, ticket transactions, orders, and voucher validations.
- **RSA Encryption:** Ensures secure transmission of data through cryptographic signatures. The RSA algorithm, coupled with SHA-256 hashing, is used to verify the integrity and authenticity of each transaction.
- **NFC and QR Codes:** NFC or QR codes are used for the transfer of ticket and voucher data between the customer app and the validation terminals.
- **Google's ML Vision Kit:** Both terminals make use of Google's ML Vision Kit (Google, n.d.) to read the QR Codes.
- **Volley:** Volley (Google, n.d.) is an HTTP library that makes networking for Android apps easier and most importantly, faster.

Data schemas

Server and Customer App Databases



We used SQLite as a database for the server. We created some dummy data for events (*Server/database/data/events.csv* & *Server/database/data/event_images/* for the event images) and products (*Server/database/data/products.sql)*. Both files are read and the information is inserted when running the server.

Applications operations

Customer App Features

- Customers can register their data (name, tax number, and credit card information) securely upon first use. The app generates a Cryptographic Key Pair which is stored in the Android Keystore.
- Users can browse upcoming performances, with details such as show dates and prices fetched in real-time from the server.
- The app allows customers to purchase up to four tickets per show (at a time). After selection, transaction details are signed cryptographically, sent to the

server for proper validation and processed using the user's stored credit card information.

- Upon purchase of tickets, customers can navigate to the Tickets Screen where a list of events for which tickets have been bought and when choosing one event, they can generate a QR Code to validate the tickets when entering the show.
- Vouchers for free cafeteria items or discounts are automatically generated in the server upon ticket purchases and stored within the app (One Free Product voucher per ticket plus 1 Voucher for a 5% discount any time the total amount of tickets bought reaches a multiple of 200€. Users can view and manage their vouchers, choosing to apply them to cafeteria orders as needed (at a maximum of 2 per order).
- Customers can compose their cafeteria orders through the app, applying vouchers to receive discounts or free items. The order and voucher details are transmitted to the cafeteria terminal for processing only after generating the QR code.
- Users can view past transactions for tickets and cafeteria orders, providing a digital receipt that includes detailed purchase information.

Ticket Validation Terminal Features

- This terminal reads ticket data from the customer app via QR codes. It checks the validity of tickets against the server database, marking them as used upon successful validation.
- The terminal displays the validation status of each ticket clearly, allowing gate attendants to manage entries efficiently.

Cafeteria Order Terminal Features

- Receives and processes orders sent from the customer app. It checks the validity of applied vouchers and calculates the final price, considering any discounts when the client scans the QR code that was generated previously.
- Shows detailed order information including the products ordered, vouchers applied, and the final price. Orders are queued for preparation, with the order number displayed for customer pickup.

Security features

1. All data transmitted between the customer app, terminals, and server is encrypted and signed using RSA keys. This ensures the integrity and confidentiality of user data and transactions.

Navigation Map

Overall Navigation Structure

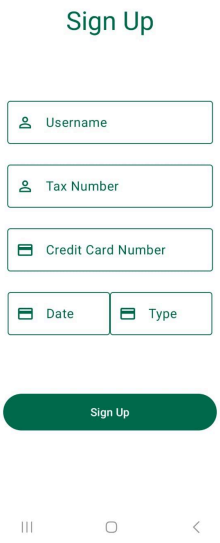
- **Registration Screen:** accessible upon first launching the app, where users input their personal information, including name, tax number, and payment details.
- **Home Screen:** displays a list of upcoming performances.
- **Event Details Screen:** Each event screen includes the date, and price per ticket, together with an option to purchase tickets (up to 4 per purchase).
- **Tickets Screen:** Displays a list of events for which tickets have been bought.
- **Event Tickets Screen:** List of ticket cards with the date of the event, price, and seat. QR codes can be generated per ticket or by choosing a set of 4.
- **Orders Screen:** List the orders made by the customer.
- **Add Order Screen:** Allows users to compose their order by selecting items from the menu. Users can apply vouchers directly on this screen, with the

system automatically updating the sub-total of the order. Orders are stored locally.

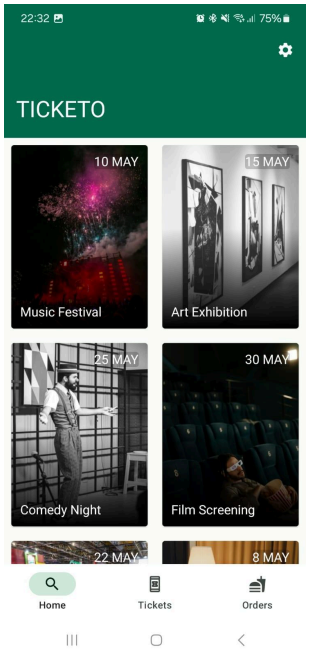
- **Order Details Screen:** Customers can see the products and vouchers associated with the order, with the option to generate a QR code to present in the Cafeteria Order Terminal. After validation, the orders are deleted from the local storage.
- **Settings Screen:** Shows customer information and options to navigate to the Past Purchases Screen and the Past Orders Screen.
- **Past Purchase Screen:** accessed from the Settings Screen. Users can see the details of past purchases (purchase date, total price, tickets and vouchers associated and customer details)
- **Past Orders Screen**: shows a detailed log of all past orders. Each entry includes detailed information such as the date, products and vouchers associated, and total money spent.

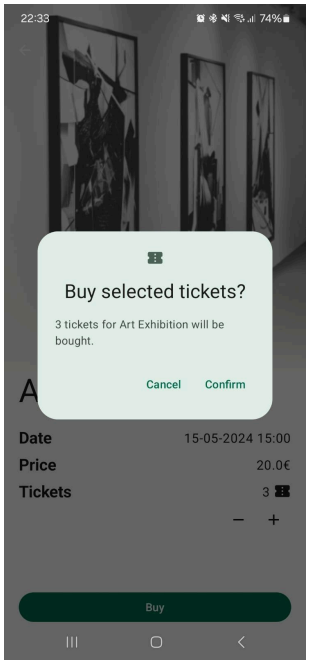
Screen Capture Sequences

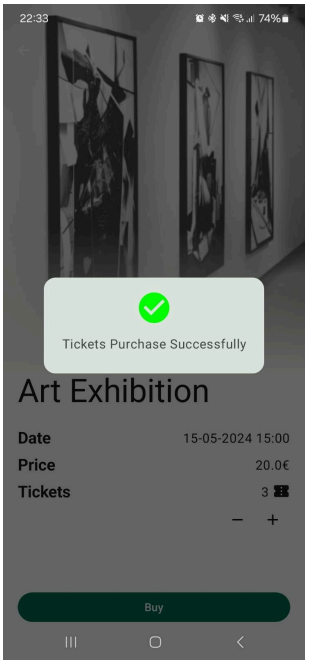
- **User Registration Flow**



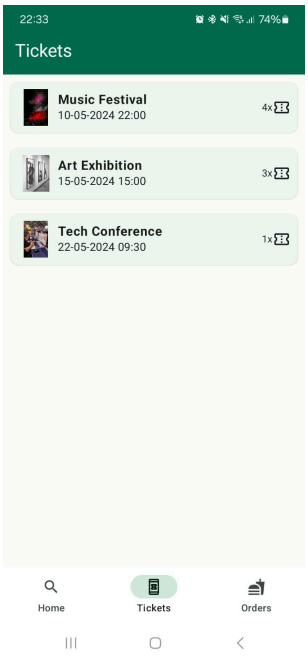
- **Tickets Buying Flow**

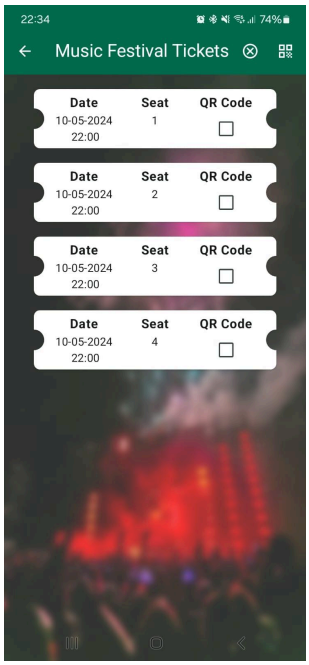


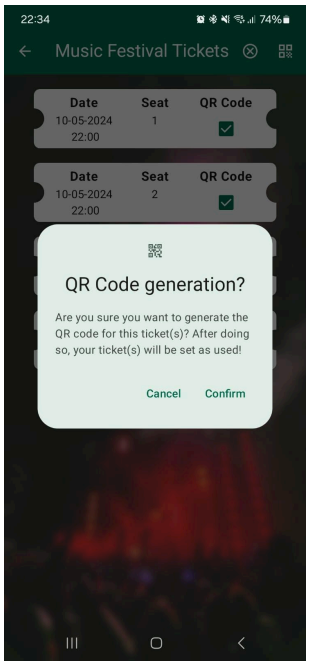


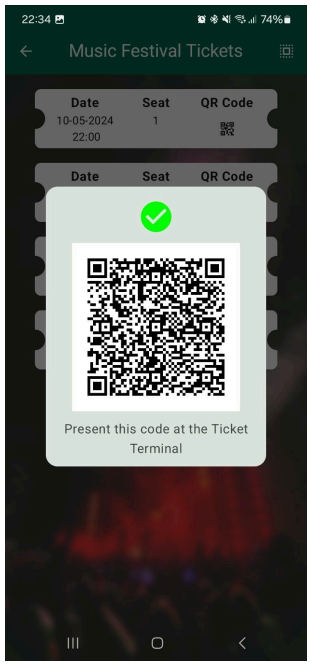


- **Tickets Validation Flow**

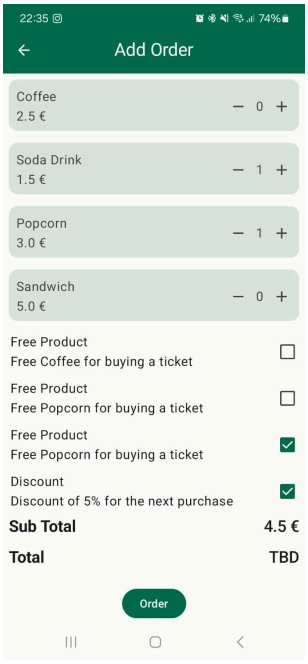


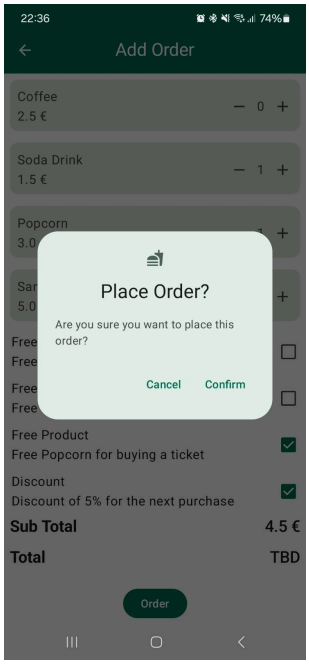


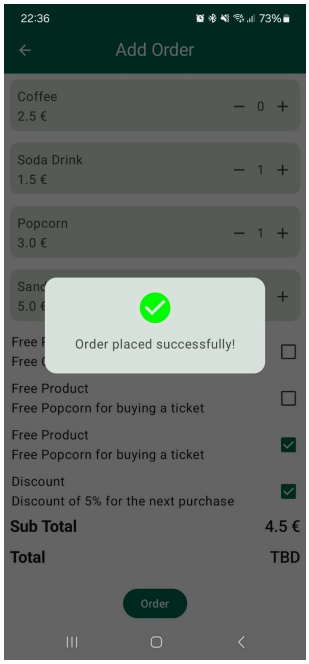




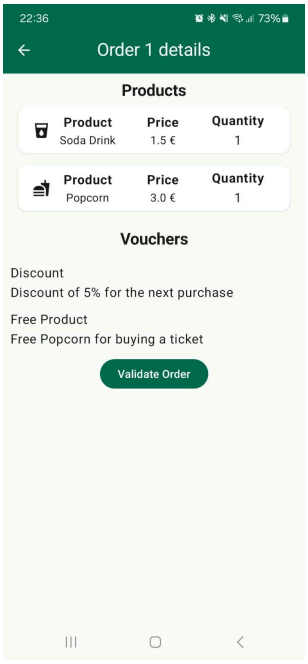
- **Order Placement Flow**

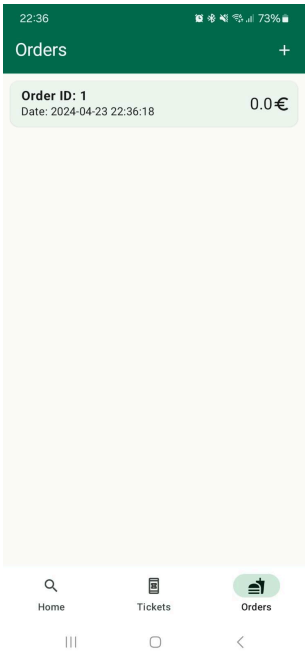


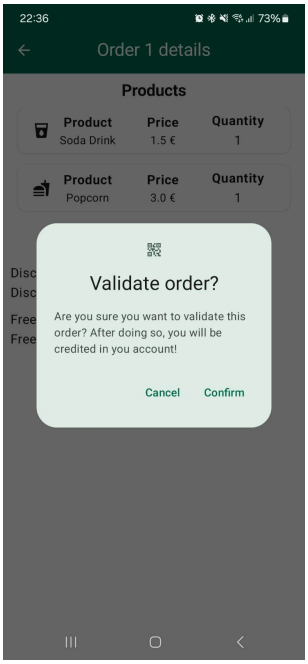


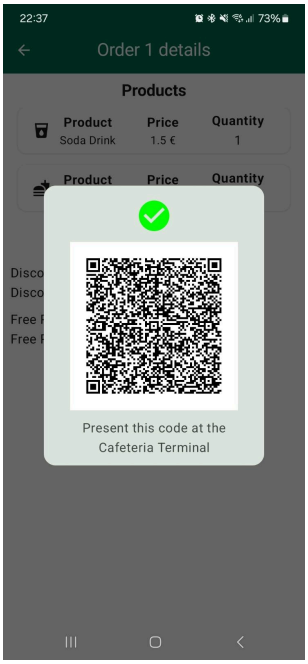


- **Order Validation Flow**

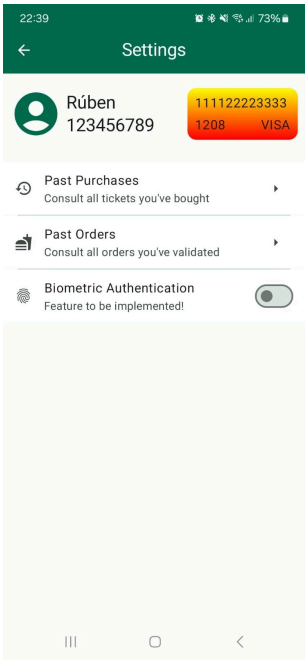


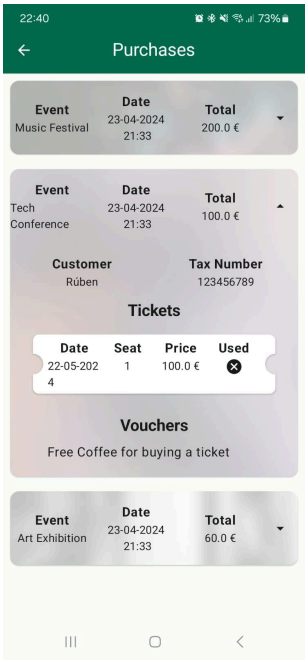






- **Consulting Past Purchases Flow**





User Guide: How to use our application?

1. As of now, the app only works smoothly when the server is running. To do so, follow the following steps:
    a. Install Python and Pip if you don't have it on your machine. We recommend creating a virtual environment.
    b. Navigate to the *server* directory (where the *app.py* file is located) from your local terminal.
    c. If you created a virtual environment, activate it.
    d. Run $pip\ install\ -r\ requirements.txt$.
    e. Run $flask\ run$ or $flask\ run\ --host = 0.0.0.0$ in case you want to run on your physical Android device and open the server to be used in the local network. It will generate an IP (different from 127.0.0.1:5000) that must be placed in the *utils/ServerCommunication.kt* file in the Customer app.
2. To run the apps, open the respective projects on Android Studio. Beware of the previous point about the IP address. If using an emulator, make sure you are using 10.0.2.2 or the generated IP. If running on a physical device, you must create a custom IP with the previous command. After Gradle Build is completed, you can launch the app on the emulator of a physical device.

References

Google. (n.d.). *Scan barcodes with ML Kit on Android*. Google for Developers. Retrieved April 6, 2024, from https://developers.google.com/ml-kit/vision/barcode-scanning/android

Google. (n.d.). *Volley overview | Volley*. Google. Retrieved April, 2024, from https://google.github.io/volley/

Google for Developers. (n.d.). *Jetpack Compose UI App Development Toolkit*. Android Developers. Retrieved April, 2024, from https://developer.android.com/develop/ui/compose

Pallets. (n.d.). *Flask*. Welcome to Flask — Flask Documentation (3.0.x). Retrieved April, 2024, from https://flask.palletsprojects.com/en/3.0.x/