M.EIC 2023/2024 CPM - 1st Assignment

Team

Group 3	
Anete Medina Pereira	202008856
João António Semedo Pereira	202007145
Mariana Solange Monteiro Rocha	202004656

Index

Overview

TikTek is a mobile app created on a project regarding the Mobile Computing course. Its main functionalities include buying event tickets and ordering from the theater cafeteria, offering an immersive experience for customers.

Development Process

The development of the TikTek project followed a structured approach, progressing through key stages to ensure the successful delivery of the application.

Mock-ups on Figma

The project began with the creation of interactive mock-ups using Figma, providing a visual representation of the application's user interface. You can access the Figma mock-ups here

Frontend and Backend Implementation

Following the mock-ups creation, we then started developing the frontend. This phase involved translating the visual design elements from Figma into functional frontend components. Each page and feature outlined in the mock-ups was implemented iteratively, allowing for easier prototyping.

Simultaneously, backend development and server setup were initiated to support the frontend implementation. The backend infrastructure was designed to handle data storage, processing, and user authentication, ensuring seamless communication between the client-side application and the server.

Iterative Improvement

Throughout the development process, iterative improvements were made to both frontend and backend components based on feedback using the app and evolving requirements, ensuring a user-friendly and robust application.

Architecture

A solid architecture is essential in mobile app development to make it less prone to errors, ensuring flexibility, scalability and maintainability. Well structured components promote modularity, simplifying app development.

To achieve that, the architecture used in the development of this app follows principles suggested in the Google Guide to App Architecture.

Our app is mostly divided into the following components:

Screen

What the user sees and interacts with on their device. Contains all the visual interface definition and structure and components such as buttons, text fields, etc.

View Model

The view model acts as a middleman between the UI and the data. Contains the

business logic required to transform raw data into a appropriate format to be presented in the UI. By acting as a middleman, it provides a **separation of concerns** which increases app modularity.

Use Case

Optional layer that contains reusable business logic. It is used when appropriate to reduce code duplication and increase code reusability.

Repositories

Repositories provide a easy way to access data. They contain methods for fetching, updating and deleting data. Their usage allows us to abstract the data sources, making it easier to switch between different data sources. They can also use network and local data sources to provide local first data access, even when offline.

Data Sources

Data sources are the data providers for the app. They include databases and network REST APIs. Data sources are the data providers that repositories use to retrieve and store data. In the context of our app, since it needs to communicate with a server, but also needs to work offline two kinds of data sources are being used: network data sources and local data sources.

Network

Data retrieved from any other source that is not the user's device. Allows real-time updates and access of information that is not locally stored.

Local

Data stored on the user's device. This kind of data source enables offline access and faster retrieval.

Work Manager

Work Manager allows us to schedule tasks that run in the background, not necessarily tied to any of the app's components. It is used to perform synchronization tasks with the server, but also to delete personal information when the user logs out.

Model

Models represent the data that is being used in the app. They give us type safety

and help us to avoid errors when working with data.

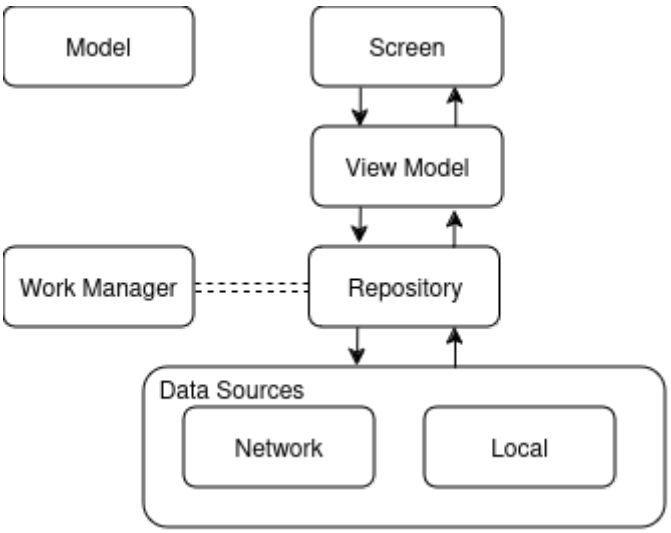


Figure 1 - TikTek Architecture

Modularization

Because multiple apps were developed in parallel, they were divided into several modules to make it easier to manage, maintain, and reuse code.

The modules fall into three main categories:

App Modules

The app modules are the main modules that contain the code specific to each app/service.

For example, the `app:main` module contains the code for the main app, while the `app:backend` module contains the code for the backend server.

These modules are:

- **app:backend**: Contains the code for the backend server.
- **app:cafeteria**: Contains the code for the cafeteria terminal.
- **app:main**: Contains the code for the main app.
- **app:tickets**: Contains the code for the ticket terminal.

Feature Modules

The feature modules contain the code for the different features of the main app. Each feature module contains the code for a different feature, and these are totally independent from each other. This way, if a feature is not needed, it can be easily removed without affecting the rest of the app.

These modules are:

- **feature:auth**: Contains the code for the authentication feature.
- **feature:cafeteria**: Contains the code for the cafeteria feature.
- **feature:events**: Contains the code for the events feature.
- **feature:profile**: Contains the code for the profile feature.
- **feature:tickets**: Contains the code for the tickets feature.

Core Modules

The core modules contain the code that can be shared between the different modules. This way, the code is not duplicated, and it is easier to maintain and update.

These modules are:

- **core:app**: Contains the code that is shared between the different app modules, such as helpers for Applications and Activities.
- **core:data**: Contains the code for the different data repositories used throughout the apps.
- **core:database**: Contains the code for the database used by the backend server.
- **core:domain**: Contains the code for the different use cases used throughout the apps.
- **core:local**: Contains the code for the different local data sources used throughout the apps.
- **core:model**: Contains the code for the different models used throughout the apps.
- **core:network**: Contains the code for the different network data sources used throughout the apps.
- **core:ui**: Contains the code for the common UI components used throughout the apps.

Build Logic

A separate module, `build-logic`, contains the common build logic used by all the modules, such as convention plugins, which standardize the build process across all the modules, and custom tasks, such as the port forwarding task.

Libraries Used

The following libraries when developing the app:

- **Akkurate**: A kotlin library for data validation. Used to validate the forms client-side and to validate the requests server-side.
- **AndroidX Compose**: A modern toolkit for building native Android UI. Used to build the UI of the app.
- **AndroidX DataStore**: A modern data storage solution that allows the app to store key-value pairs or typed objects. Used to store the user's authentication token, profile, and cart locally.
- **AndroidX Lifecycle**: A library that provides lifecycle-aware components, such as View Models. Used to observe data changes in the app.
- **AndroidX Navigation**: A library that provides a framework for navigating between different screens in the app.
- **AndroidX Room**: A library that provides an abstraction layer over SQLite. Used to store data locally in the app.
- **AndroidX SplashScreen**: A library that provides a splash screen API for Android. Used to show a splash screen when the app is launched.
- **AndroidX Work**: A library that provides a way to schedule tasks that run in the background. Used to synchronize data with the server and delete personal information when the user logs out.
- **BCrypt**: A library that provides a way to hash passwords using the BCrypt algorithm. Used to hash passwords before storing them in the database.
- **Coil**: A modern image loading library for Android. Used to load images in the app.
- **Compose Destinations**: A wrapper around AndroidX Compose and AndroidX Navigation that uses annotations and code generation to simplify navigation in Android apps. Used to define destinations in the app.
- **Dagger**: A dependency injection library. Used to inject dependencies into the app.
- **Exposed**: A Kotlin ORM library. Used to interact with the database in the backend server.
- **Faker**: A library that generates fake data. Used to generate fake events for testing purposes.
- **H2**: An in-memory database. Used to store data in the backend server.
- **Hilt**: A dependency injection library for Android. Uses Dagger under the hood. Used to inject dependencies into the app.
- **KotlinX DateTime**: A library that provides a way to work with dates and times in Kotlin. Used to work with dates and times in the app.
- **KotlinX Serialization**: A library that provides a way to serialize and deserialize data in Kotlin. Used to serialize and deserialize data for communication between apps and the server.

- **KTLint**: A static code analysis tool for Kotlin. Used to enforce coding standards in the app.
- **Ktor**: A Kotlin networking library. Used to implement the server in the backend server.
- **Retrofit**: A type-safe HTTP client for Android. Used to make network requests in the app.
- **Timber**: A logging library for Android. Used to log messages in the app.
- **ZXing**: A barcode scanning library for Android. Used to scan and generate QR codes in the app.

Data Schemas

Server Database

☕ Cafeteria Item

Cafeteria Items are stored in the database with a string ID, which is the primary key, the item name (Name), price (Price) stored as an integer (because the price is stored in the database as cents of euro and converted for euros every time it is shown to the user), and an image URL (Image Url) to represent the item visually.

Name	Type
ID*	String
Name	String
Price	Integer
Image Url	String

📋 Order

The Order schema represents an order made by a customer. Each order has a unique identifier (ID) and a date (Date) indicating when the order was made.

Attribute	Type
ID*	String
Date	Date

🏷️ Voucher

The Voucher schema represents a voucher that can be redeemed by a user for

discounts or special offers. Each voucher has a unique identifier (ID), a discount amount (Discount), an optional associated item (Item), a user who owns the voucher (User), and an optional associated order (Order).

When users spend 200€ on tickets they receive a 5% discount voucher and each time they buy a ticket for an event they receive a free product voucher with a random product.

Attribute	Type
ID*	String
Discount	Integer (Nullable)
Item	Item (Nullable)
User	User
Order	Order (Nullable)

🎭 Event

The Event schema represents an event available in the app. It includes details such as the event's unique identifier (ID), name (Name), description (Description), date (Date), start time (Start Time), end time (End Time), location (Location), optional location details (Location Details), price (Price), and an image URL (ImageUrl).

Attribute	Type
ID*	String
Name	String
Description	String
Date	Date
Start Time	Time
End Time	Time
Location	String
Location Details	String (Nullable)
Price	Integer
ImageUrl	String

🎫 Ticket

The Ticket schema represents a ticket purchased by a user for a specific event. Each ticket has a unique identifier (ID), an associated event (Event), user who purchased the ticket (User), associated seat (Seat), purchase date (Purchase Date), and optional use date (Use Date).

Attribute	Type
ID*	String
Event	Event
User	User
Seat	Seat
Purchase Date	Date
Use Date	Date (Nullable)

👥 User

The User schema represents a registered user in the system. It includes attributes such as a unique identifier (ID), name (Name), tax identification number (NIF), birthdate (Birthdate), email address (E-mail), name on credit card (Name CC), credit card number (Number CC), credit card expiration date (Expiration Date CC), and password (Password).

Attribute	Type
ID*	String
Name	String
NIF	String
Birthdate	Date
E-mail	String
Name CC	String
Number CC	String
Expiration Date CC	String
Password	String

Navigation Map

Pages highlighted in pink within the navigation map represent sections accessible via the bottom navigation bar. These sections offer easy access and, since the user is logged in, can be reached with just a single click.

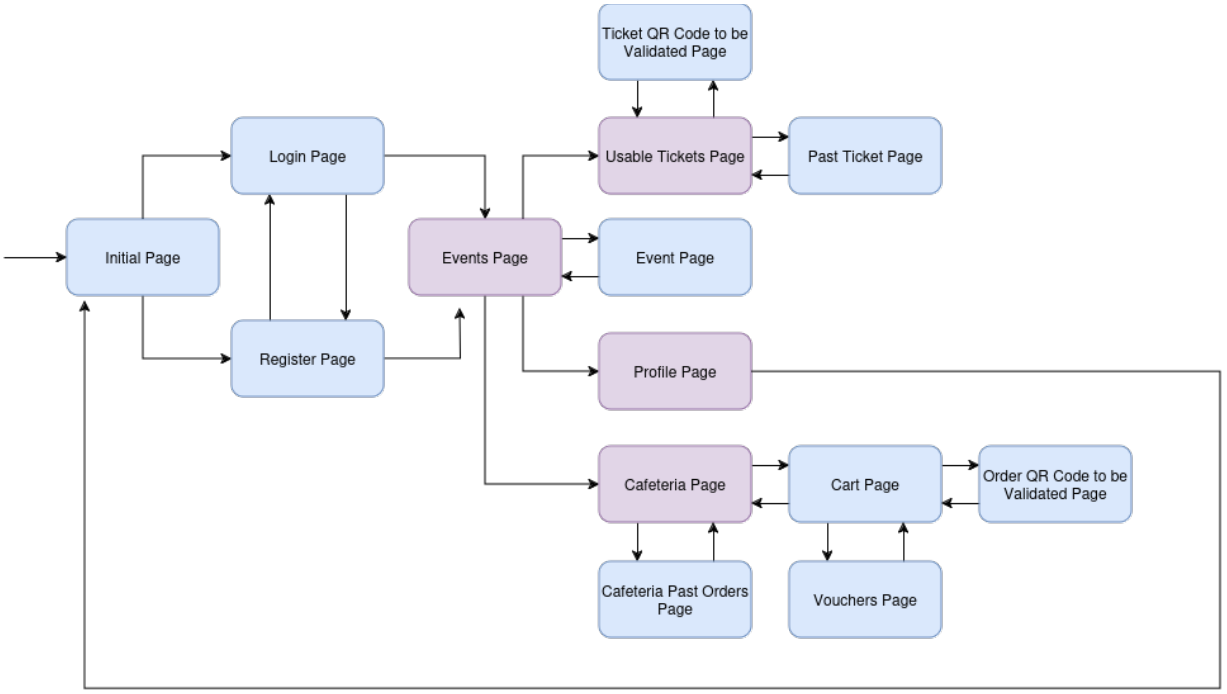


Figure 2 - TikTek Navigation Map

Security

The app authentication is done by verifying an email and password pair with the stored user info in the database. This password is hashed using the bcrypt algorithm before it's stored.

User sessions are represented using a JSON Web Token, which is signed and encrypted by the server. This JWT stores the user id for authorization, and the user public key.

This public key is used to verify the request body signature, which is sent with every request, and signed by the app using the user's private key. This key pair is generated each time the user signs in to a new app, and is stored in the android keystore.

The terminal apps do not use any authentication, as they are assumed to be in a protected network when deployed correctly, so no attacks are feasible. In this vein, no encryption is done, because in a correct deployment HTTPS/TLS should be used for encryption.

Features

Main Features

Authentication

When users open the app for the first time, they are directed to the "Initial Page" (first from left to right) featuring two primary options: Log In or Sign Up. Once logged in, regardless of whether the app is closed or not, the default landing page becomes the "Events Page".

The users can create an account if they don't have one or log in in a previously registered account as shown in the screenshots below.

Features

- [x] Register account
- [x] Log into account
- [x] Mathematical Credit card validation upon creation
- [x] Mathematical SSN validation upon creation
- [x] User needs to be 13 years-old or older to register an account

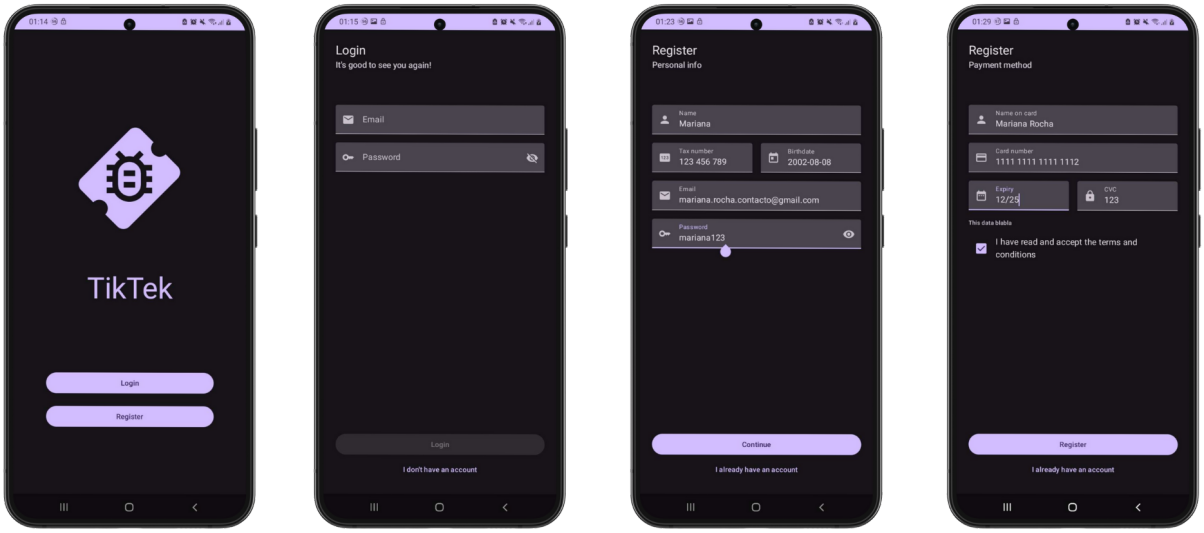


Figure 3 - Authentication Pages

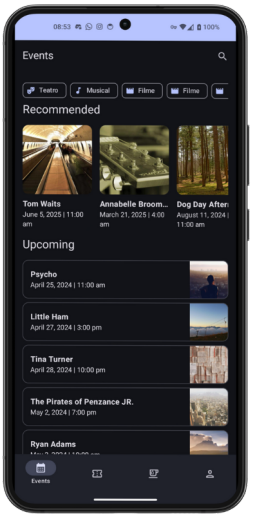
Events

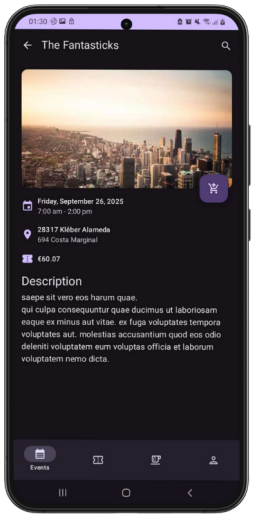
When the users log in, as referred above, they are redirected to the events page. There they can check the upcoming events, buy tickets and collect information

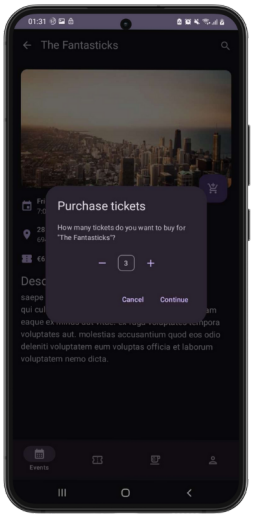
about specific events as shown in the pictures below. When an user attempts to finish a purchase they have to confirm that they are the owners of the phone by using biometric authentication.

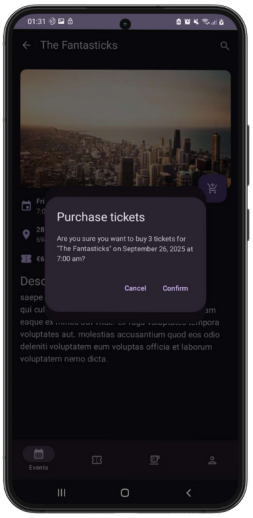
Features

- [x] Check the next events taking place
- [x] Check information about a specific event (date, time, price, description)
- [x] Buy tickets to upcoming events
- [x] Check my event tickets that are still available to be used
- [x] Check my event tickets history
- [x] Generate QR code to confirm purchase on the terminal









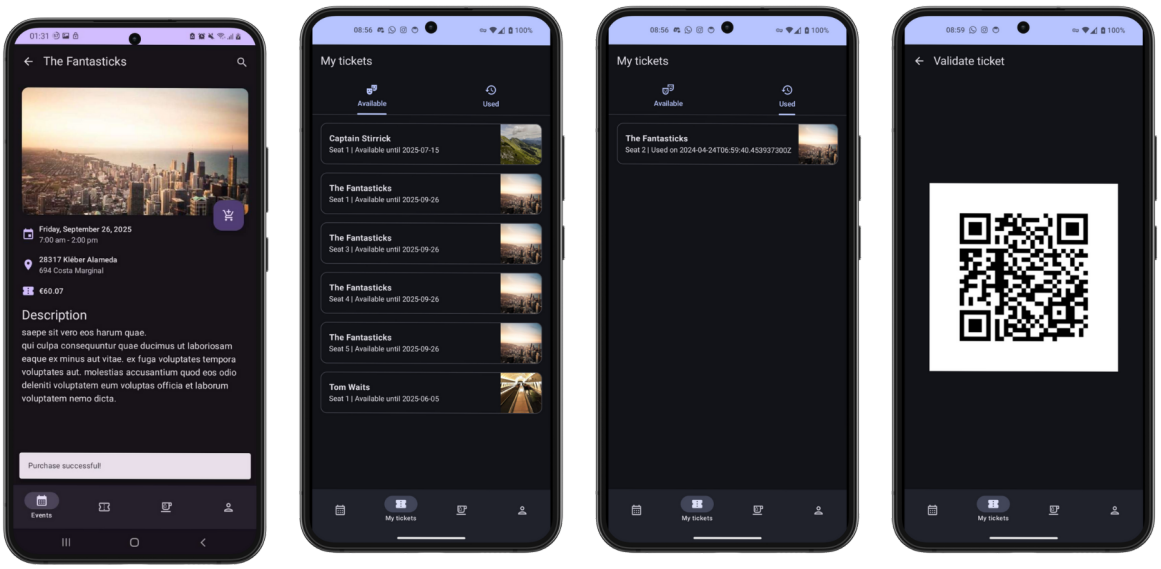


Figure 4 - Events' Pages

Cafeteria

Besides being able to buy and check information about upcoming event, users are also allowed to buy from the local theatre cafeteria using the app. When an user attempts to finish a purchase, just like in events tickets, they have to confirm that they are the owners of the phone by using biometric authentication. After a purchase is made, a QR code is generated to be validated on the terminal. It contains all the products bought and their quantity, vouchers used and the price before and after the use of the vouchers.

Features

- [x] Check items available in the cafeteria menu
- [x] Check if cafeteria is open or closed
- [x] Add items to cart
- [x] Remove items from cart
- [x] Change items quantity in cart
- [x] Buy cafeteria order
- [x] Cancel cafeteria order
- [x] Generate QR code to confirm purchase on the terminal
- [x] Check my cafeteria orders that are still available to be used
- [x] Check my cafeteria orders history and receipts

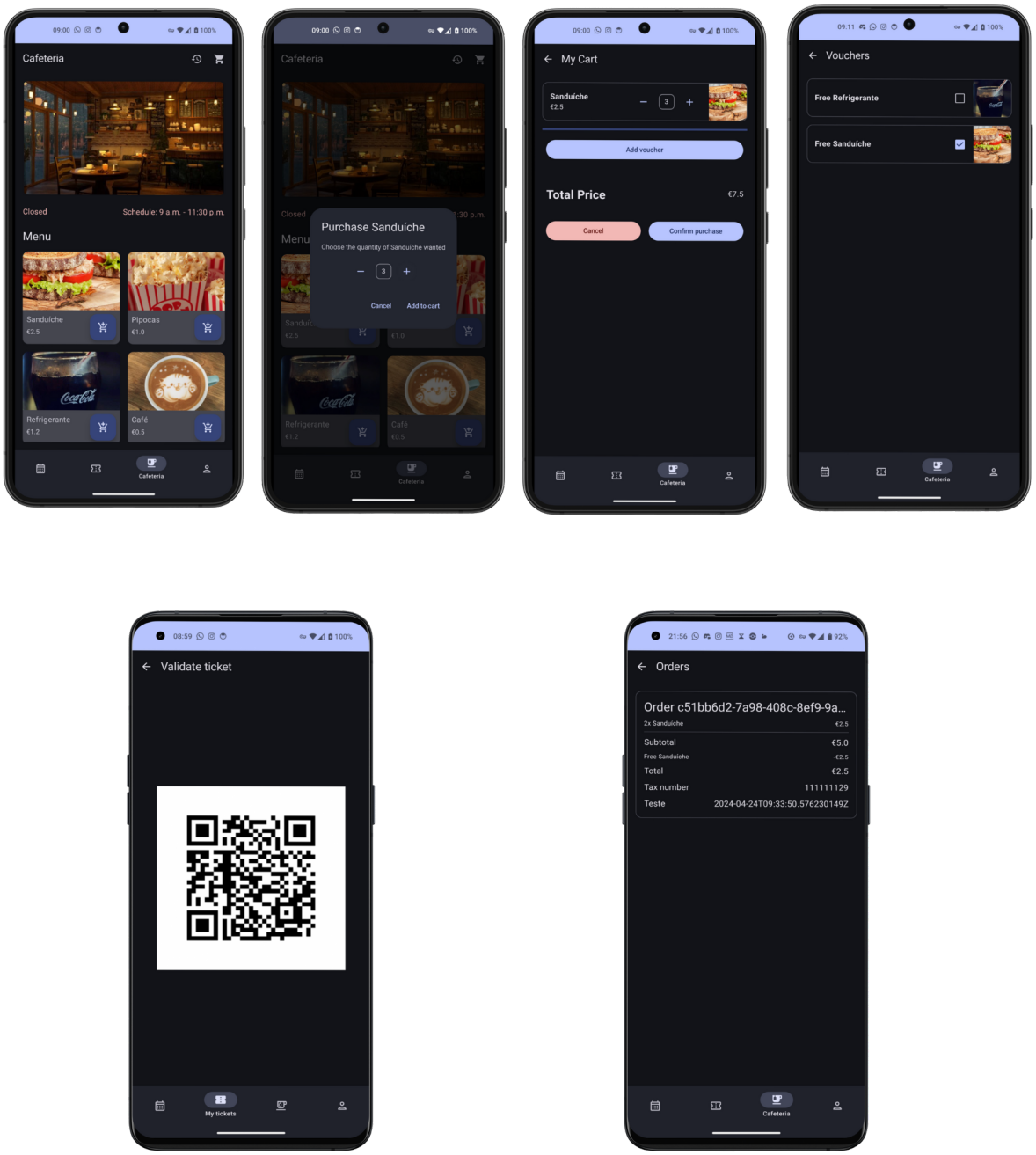


Figure 5 - Cafeteria's Pages

Profile

Users can access their personal page to update their personal information or their payment details (update outdated credit card). In this page users can log out of the app if they want.

Features

- [x] Logout
- [x] Change personal details and payment information
- [x] Mathematical Credit card validation upon change

- [x] Mathematical SSN validation upon change

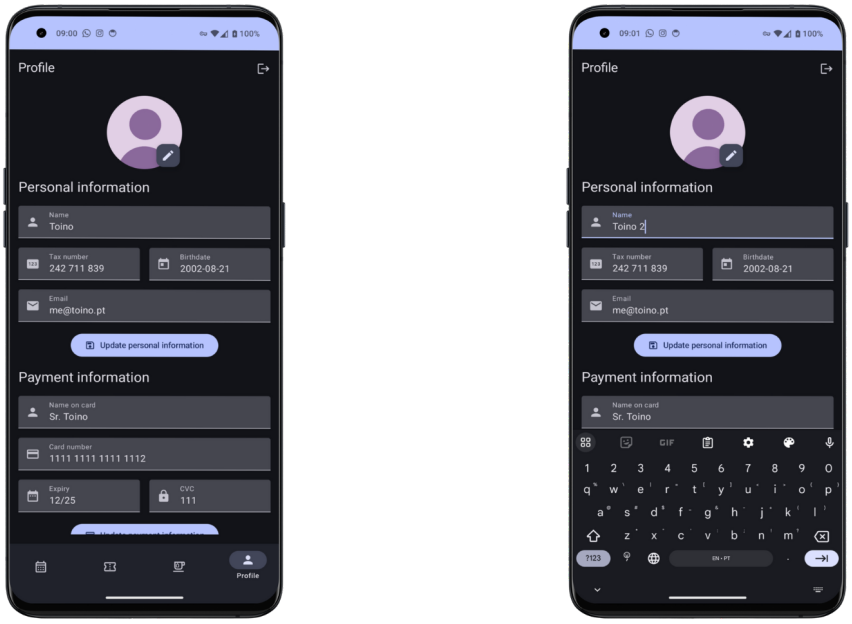


Figure 6 - Profile's Pages

Ticket Terminal

The ticket terminal allows customers to validate their purchased tickets. Users simply scan the QR code generated by the main app on their device using the terminal's camera. The terminal then sends the ticket information to the server for validation and displays the validation status.

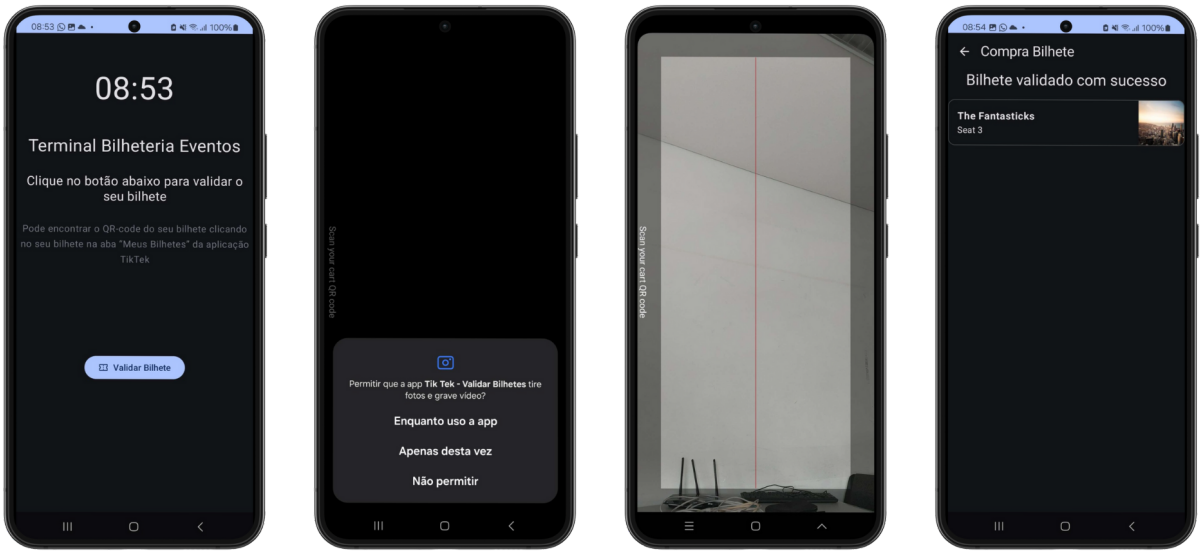


Figure 7 - Ticket Terminal Pages

Cafeteria Terminal

The cafeteria terminal facilitates ordering and payment for cafeteria items. Customers scan the QR code generated by the main app on their device using the terminal's camera. The terminal sends the cart information to the server for validation and displays a confirmation screen with the purchased items and total price.

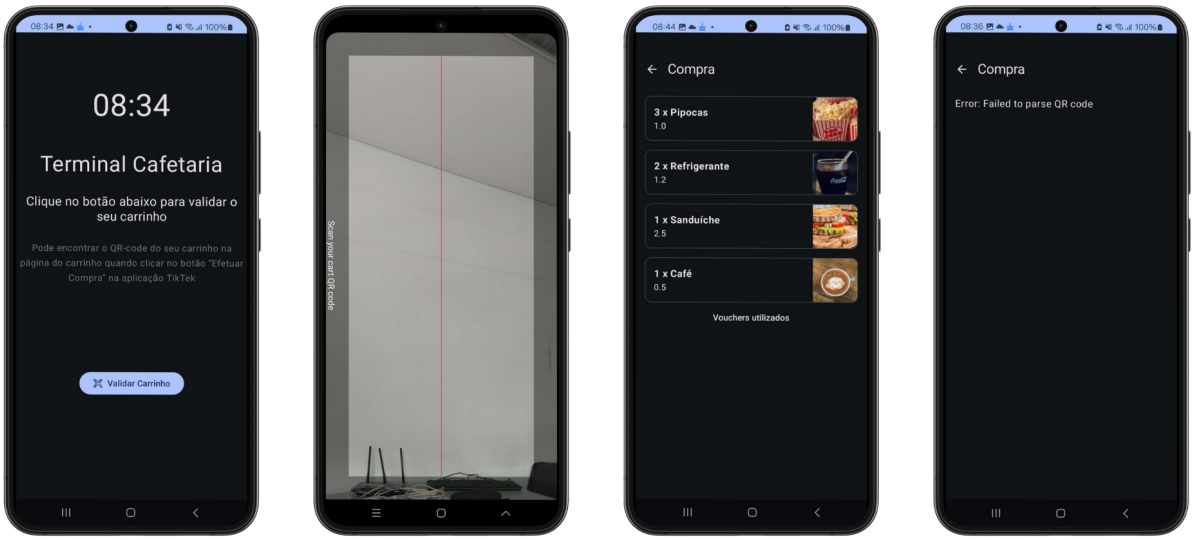


Figure 8 - Cafeteria Terminal Pages

Other Features

Dark and Light Mode

We used Material Theme 3 for the development of our app UI. Depending on the default theme of the user's phone the app will appear on dark or light mode.

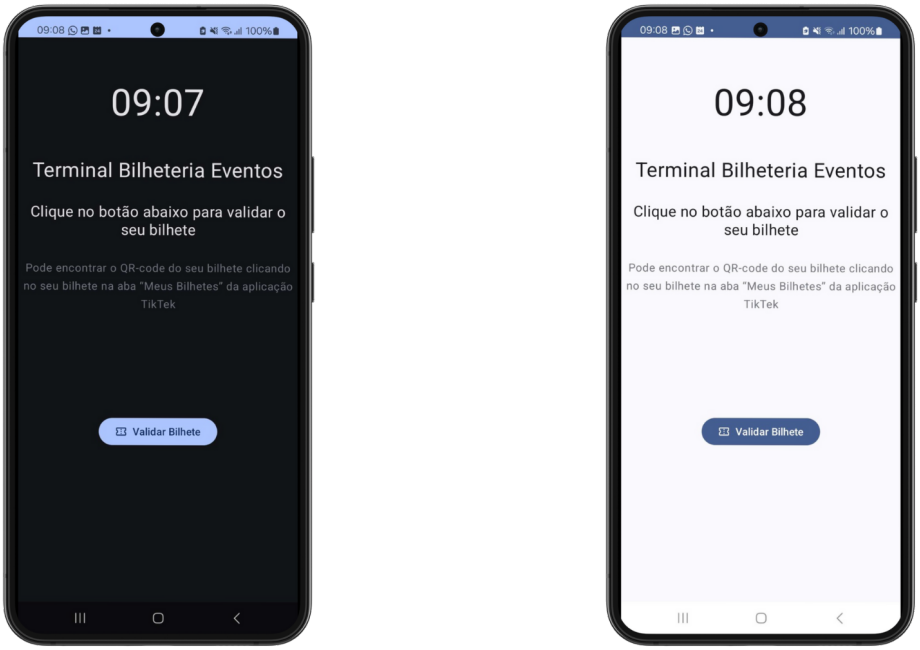


Figure 9 - Dark and Light Mode Example

Available in English and Portuguese Languages

We made the app available in English and Portuguese using the resource of "strings.xml" to implement so.

Performed Scenarios Test and How to use

This section outlines some test scenarios conducted to evaluate the functionality and usability of the TikTek mobile application and the terminals. These scenarios cover key interactions such as user registration, order placement, ticket purchase, validation, and voucher redemption. Each scenario is described along with the expected outcome to ensure thorough testing of the application's features. Screenshots accompany the scenarios to provide visual clarity. This testing phase aims to assess the performance and user experience of the application.

1. User Registration

Scenario Description: This scenario tests the user registration process. **Test Steps:**

1. User opens the app and selects the "Sign Up" option.
2. User enters their personal details, including name, email, birthdate, and credit card information.
3. User submits the registration form.
4. **Expected Outcome:** The user is successfully registered, and a confirmation

message is displayed.

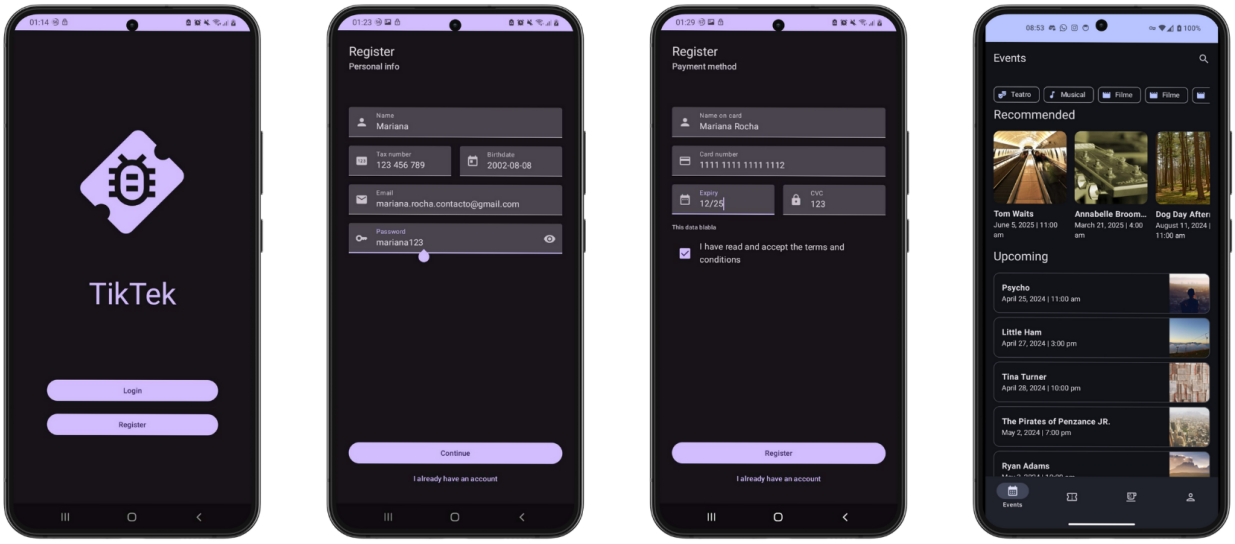


Figure 10 - User Registration Tutorial

2. Cafeteria Order

Scenario Description: This scenario tests the process of placing a cafeteria order.
Test Steps:

1. User navigates to the cafeteria menu section in the app.
2. User adds items to their cart, adjusting quantities if necessary.
3. User proceeds to checkout and confirms the order.
4. **Expected Outcome:** The cafeteria order is successfully placed, and a confirmation screen displays the ordered items and total price.

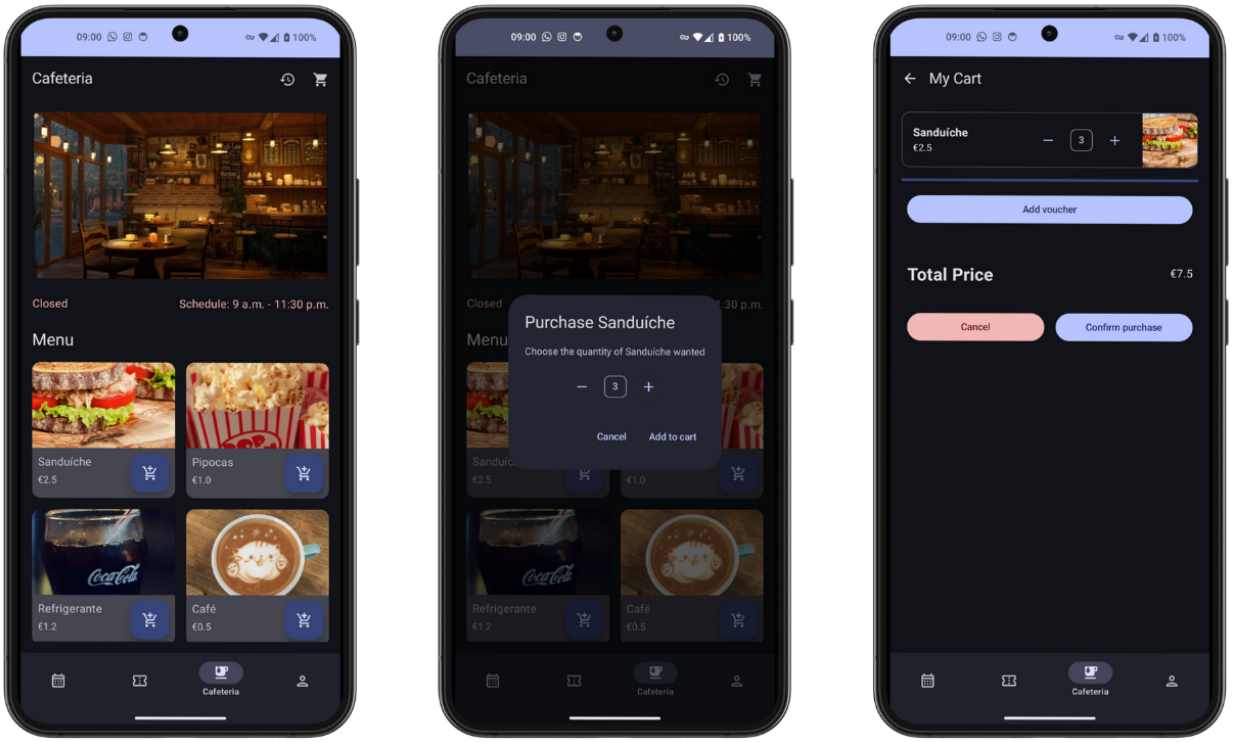
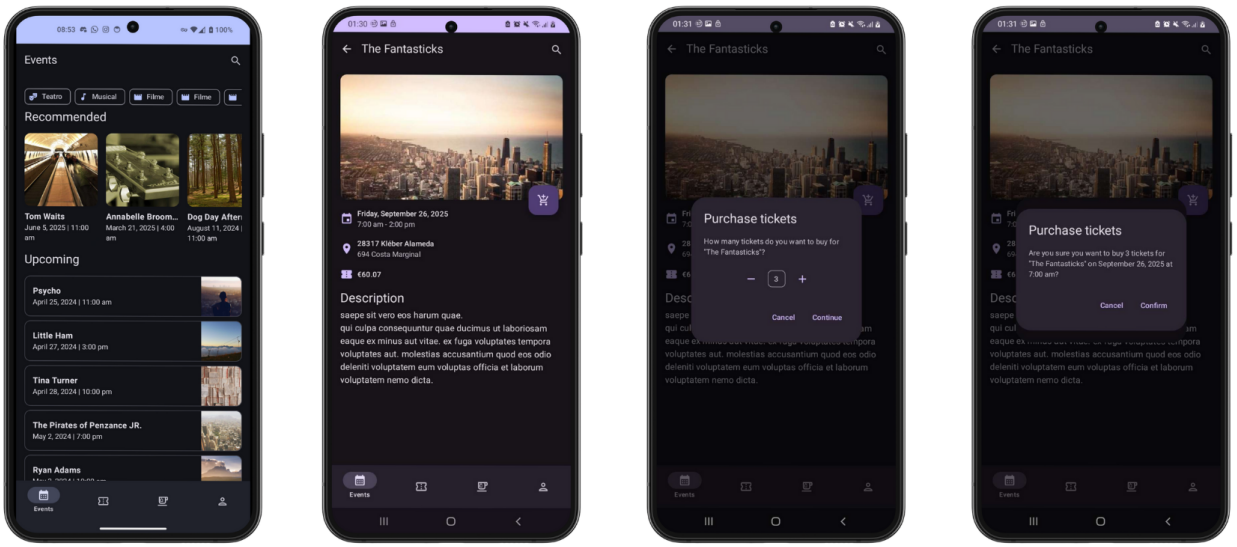


Figure 11 - Cafeteria User Tutorial

3. Ticket Purchase

Scenario Description: This scenario tests the process of purchasing tickets for an event. **Test Steps:**

1. User browses upcoming events in the app.
2. User selects an event and chooses the desired number of tickets.
3. User completes the ticket purchase process, entering payment details if necessary.
4. **Expected Outcome:** The ticket purchase is successful, and the purchased tickets are displayed in the user's account.



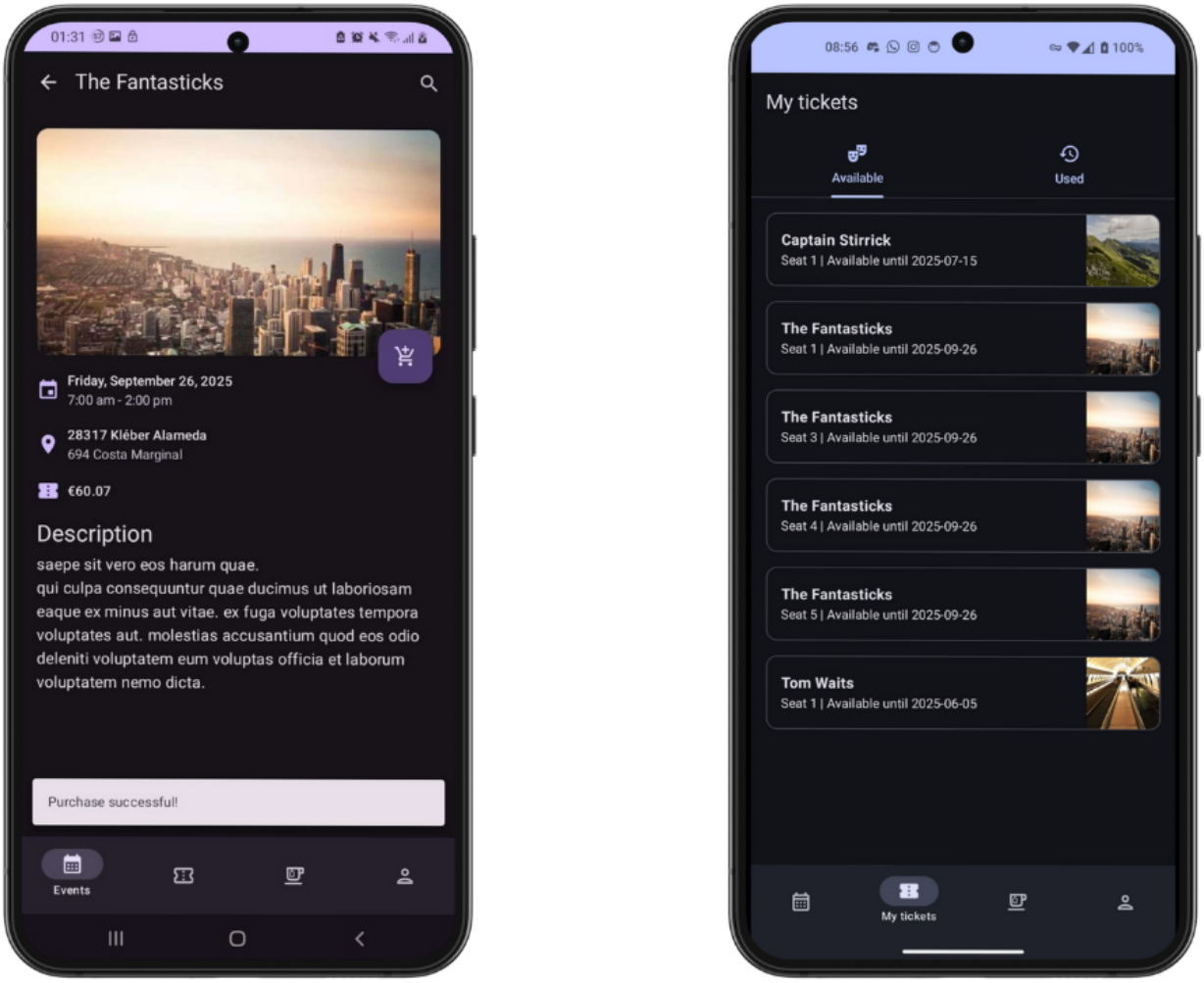


Figure 12 - Ticket Purchase Tutorial

4. Ticket Validation

Scenario Description: This scenario tests the validation of purchased tickets at the ticket terminal. **Test Steps:**

1. User approaches the ticket terminal with their smartphone.
2. User opens the app and displays the QR code for the purchased ticket in the terminal.
3. Terminal scans the QR code and sends the ticket information to the server for validation.
4. **Expected Outcome:** The ticket is successfully validated, and the terminal displays a validation confirmation.

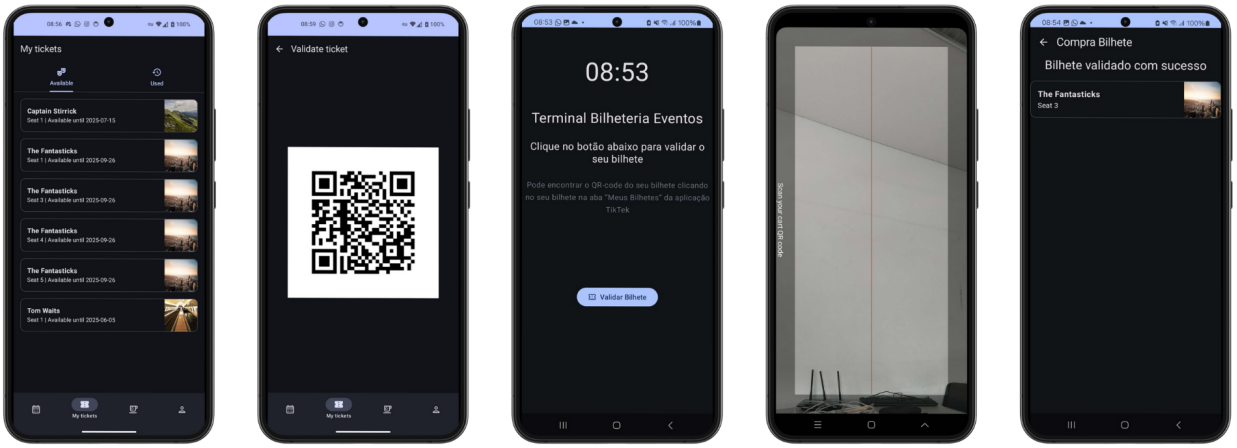


Figure 13 - Ticket Validation Tutorial

5. Voucher Redemption

Scenario Description: This scenario tests the redemption of vouchers for discounts or special offers. **Test Steps:**

1. User selects an item for purchase and proceeds to checkout.
2. User applies a voucher for a discount or free item, if available.
3. Terminal sends the order information, including the applied voucher, to the server for validation.
4. **Expected Outcome:** The voucher is successfully redeemed, and the terminal displays the adjusted total price for the order.

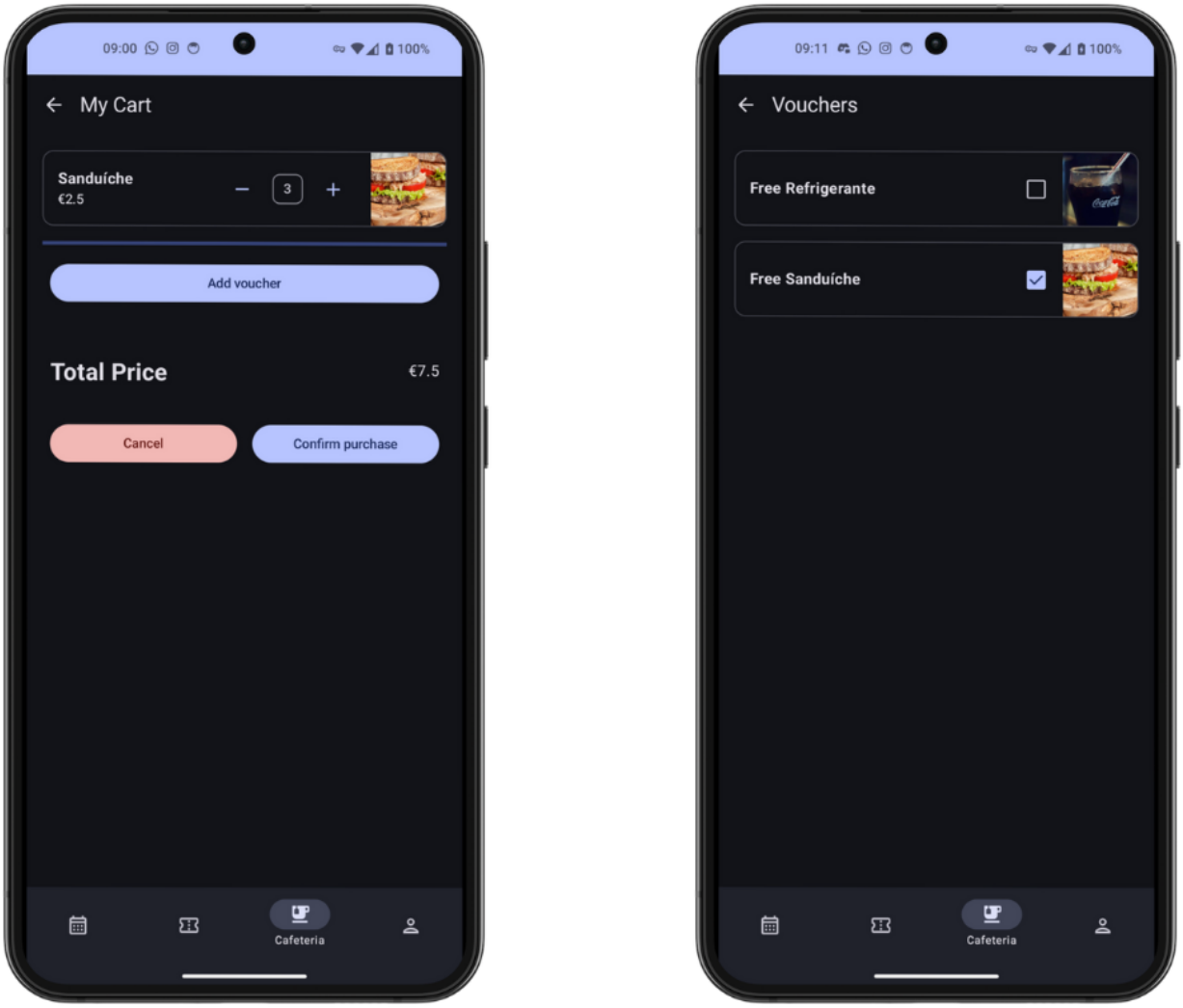


Figure 14 - Voucher Redemption Tutorial

References

1. Google Guide to App Architecture
2. Google Guide to App Modularization