BusTracker: Flutter Project

Guilherme Antunes [103600], Pedro Rasinhas [103541]
v2024-11-03

1 Application Concept

BusTracker is a mobile application designed to revolutionize the public transportation experience by providing users with real-time bus tracking, easy access to bus schedules via QR codes at bus stops, and an NFC-based ticket validation system. The primary goal is to enhance convenience, reduce wait times, and streamline the boarding process for public transport users.

1.1 Objectives

- **Real-Time Bus Tracking:** Enable users to view live bus locations on an interactive map, allowing for better journey planning and reduced waiting periods at bus stops.

- **QR Code Schedule Access**: Facilitate quick access to bus schedules and stop timings by scanning QR codes available at bus stops.
- **NFC Ticket Validation**: Implement an NFC-based system that allows users to validate their bus tickets using their smartphones or NFC-enabled cards.
- **Offline Accessibility**: Ensure essential information like bus schedules and the map tiles cached and travel history is available offline through local data persistence.
- **Connectivity Awareness**: Provide seamless functionality by detecting the user's connectivity status and adjusting features accordingly.

1.2 Technologies Used

- **GPS/Location Services/Map**: To look for stops, tracking buses, and see current user location;
- **Camera**: To scan QR codes at bus stops;
- **NFC**: For contactless ticket validation;
- **Local Persistence (Isar Database)**: To store data locally on the device;
- **Connectivity Detection**: To manage online and offline modes;
- **External Resources:**
    - **Firebase**: Used exclusively for secure user authentication, managing user credentials and login sessions.
    - **Azure (API deployment):** FastAPI-based backend deployed on Azure;
- **Data mining:** Scrapping the bus routes, stops and schedules from AveiroBus website

2 Implemented Solution

2.1 Architecture Overview

The BusTracker application is built using a client-server architecture, leveraging FastAPI for the backend and a Flutter-based mobile client. The backend API is deployed on Azure, ensuring scalability and reliability. The mobile application utilizes Isar, a local database, for offline data persistence. This setup ensures a responsive user experience both online and offline.

We also developed a second application, BusTracker Validator, which, as the name states, works as an app to validate the cards using NFC.

Backend - FastAPI on Azure:

- The API serves as the central hub for data operations, handling requests from the mobile client, such as fetching bus schedules, live tracking data, and travel history.
- SQLite is used for database interactions, providing a robust and secure way to manage data.

Database Design:

- The database is structured to manage users, travel histories, buses, routes, and stops, with well-defined relationships between entities.
- Indexing on critical columns (e.g., `firebase_uid`, `card_number`, `route_id`) enhances query performance, essential for a real-time application.

Mobile Client - Flutter with Isar:

- Isar, an efficient local database, is used for storing data locally, such as bus stops, routes, and travel history. This allows users to access important information even without an internet connection.
- The mobile client integrates GPS for real-time location services, the camera for QR code scanning, and NFC for ticket validation, fully utilizing the device's capabilities.

2.2 Detailed Components

2.2.1. Database Architecture

2.2.1.1 SQLite diagram

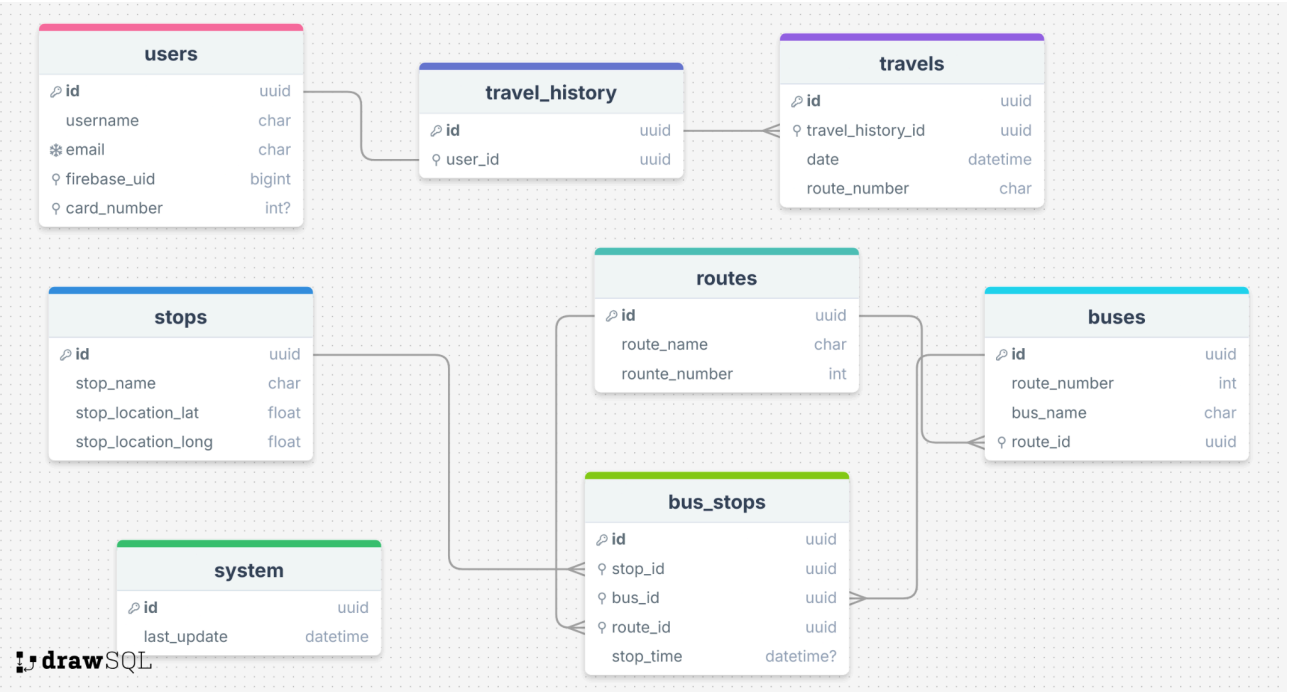


Figure 1 - Server SQLite Database Diagram

- The database schema includes tables for `users`, `travels`, `travel_history`, `stops`, `routes`, `buses`, `bus_stops`, and `system`.

2.2.1.2. ISAR diagram

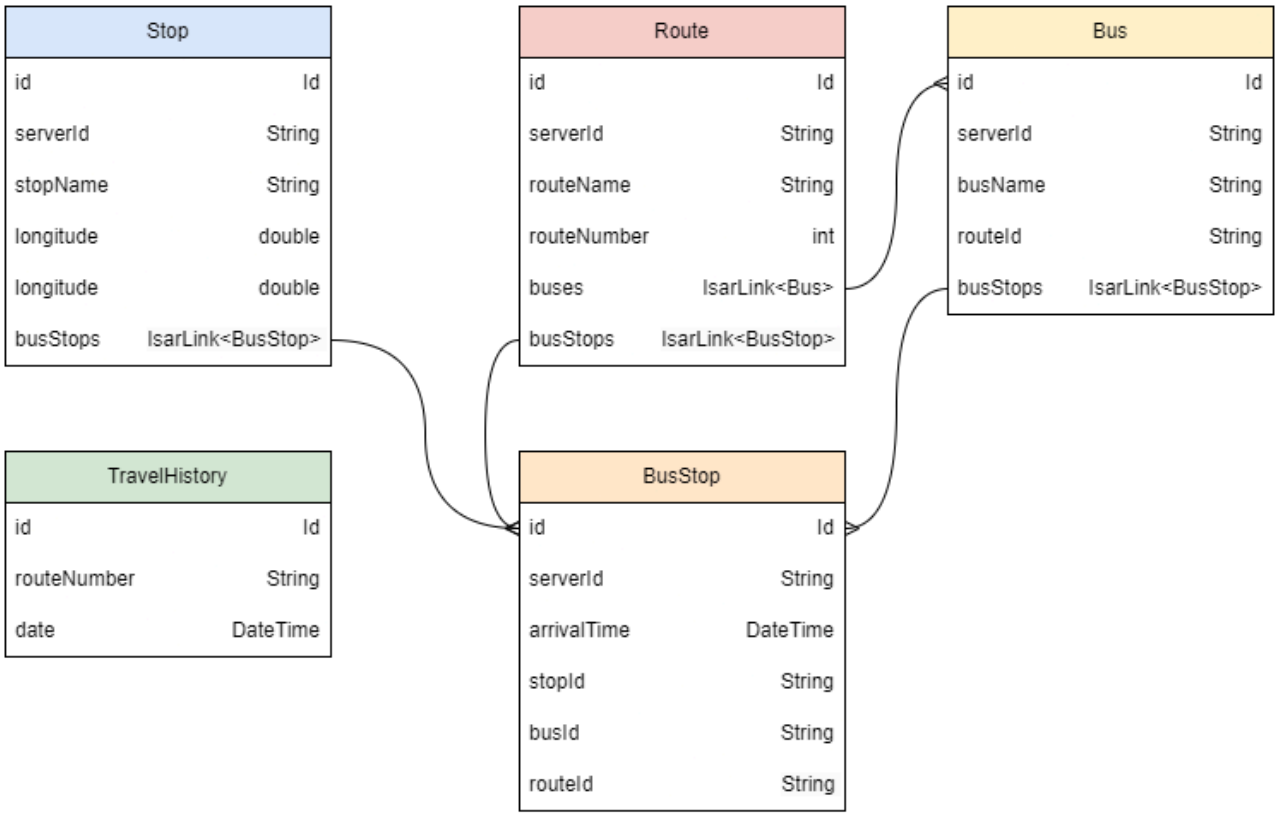


Figure 2 - Local ISAR Database Diagram

2.3 User Stories

- **(1)** As a user, I want to search for a specific bus stop using a search bar, view it on the map, and access its schedule, so that I can quickly locate the stop I need and check bus timings without manually navigating through the map.
- **(2)** As a user, I want to locate nearby bus stops and view live buses on a map, click on a stop to view its details, and access the schedule page directly, so that I can quickly find nearby transportation options, check when the next bus will arrive, and plan my journey efficiently.
- **(3)** As a user, I want to click on a live bus on the map to view the bus's route information and access detailed stop locations along that route, so that I can decide whether that specific bus and route align with my destination.
- **(4)** As a user, I want to scan a QR code at a bus stop to immediately access its schedule and view upcoming buses, including each bus's route details, so that I can decide which bus to take based on the route and estimated arrival time without needing to manually search.

- **(5)** As a user, I want to validate my ticket using NFC and have my travel history automatically stored in my profile, so that I can keep a record of my journeys and validate my ticket easily with my mobile device.
- **(6)** As a user, I want the app to store bus schedules and routes locally and automatically switch to offline mode when there's no internet connection, so that I can continue to access essential bus information and schedules, even in areas with poor connectivity.

2.4 Implemented Interactions

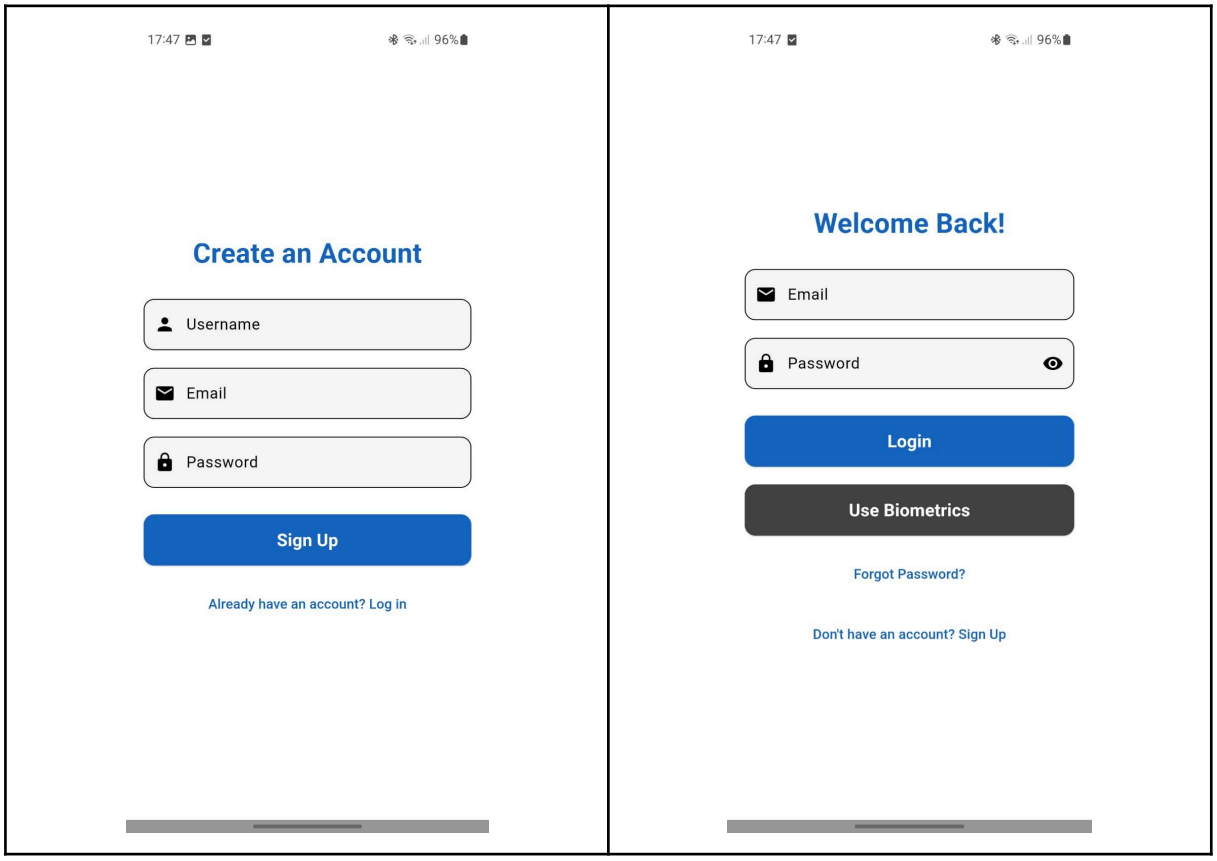


Figure 3 - Sign up and Login pages (firebase authentication)

When launching the BusTracker app you will land at the right screen of Figure 3, if it is your first time the "Use Biometrics" button will not be available since to enable it you need to successful login one time and press "Yes" when asked if you want to activate biometric authentication. Example credentials: Email = a@gmail.com, Password = Ola12345

You can also create a new account traveling to the left screen of Figure 3 by pressing "Don't have an account? Sign Up"

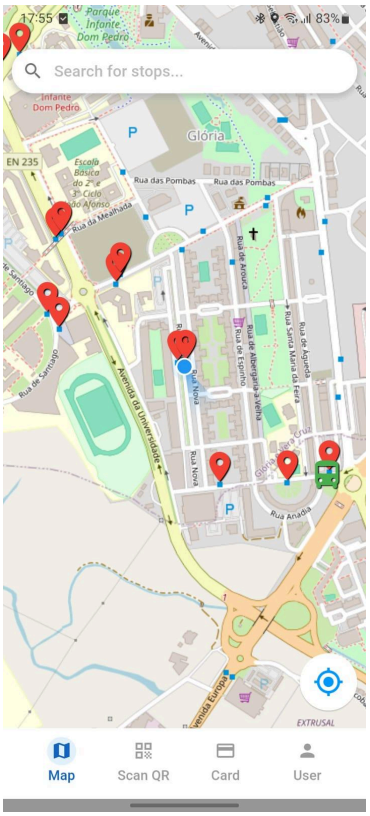
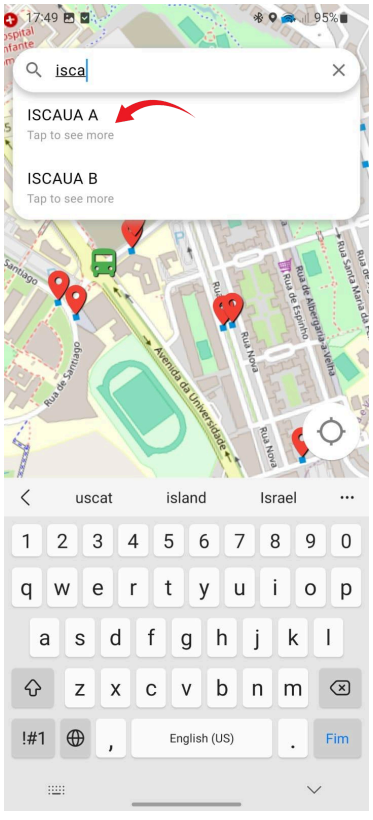


Figure 4 - Landing page



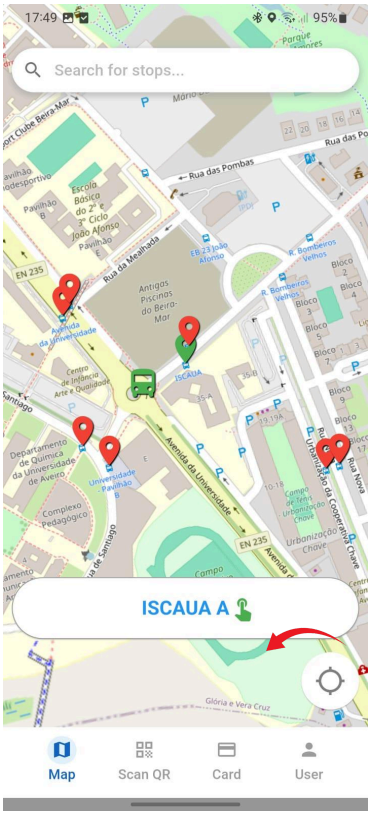


Figure 5 - Search for stops using search bar and results from search bar

After successful login, a fetch request is sent to the API to retrieve all data related to stops, buses, and bus routes, which is then stored in the persistent database. After that you will face a map landing page with a search bar on top (Figure 4). You can search a stop by its name (Figure 5 left) or you can explore the ones displayed on the map (Figure 6 right). After you press one option in the search bar or press one of those displayed in the map you will notice that a green marker and a floating button will appear (Figure 6). If you press the new floating button the application will show you the buses scheduled to pass through that stop. You can also press a moving bus to know its schedule.

If your location service is enabled you can experience other ways to view your position by clicking on the bottom-left floating circle button, which must have the blue or gray color if the location is enabled.

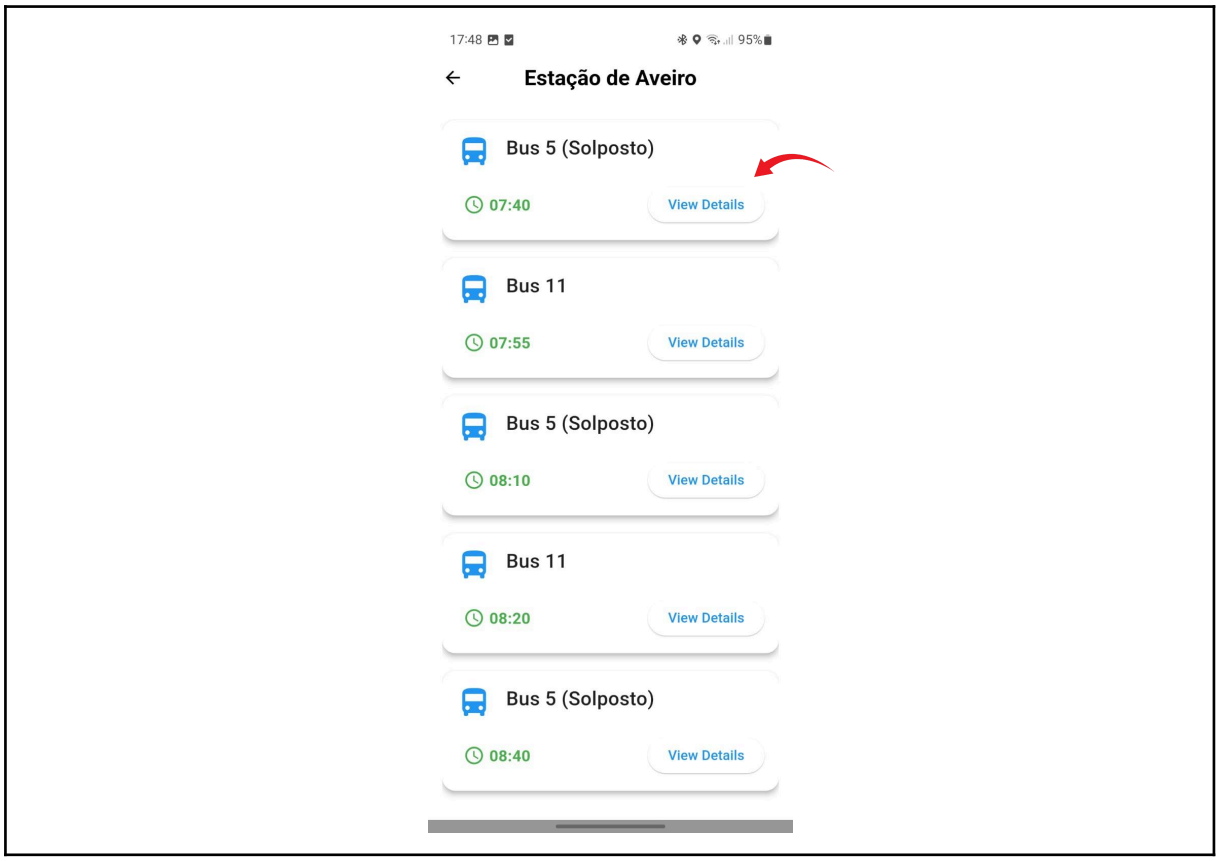


Figure 6 - Selecting the stop to see the details (Stop page)

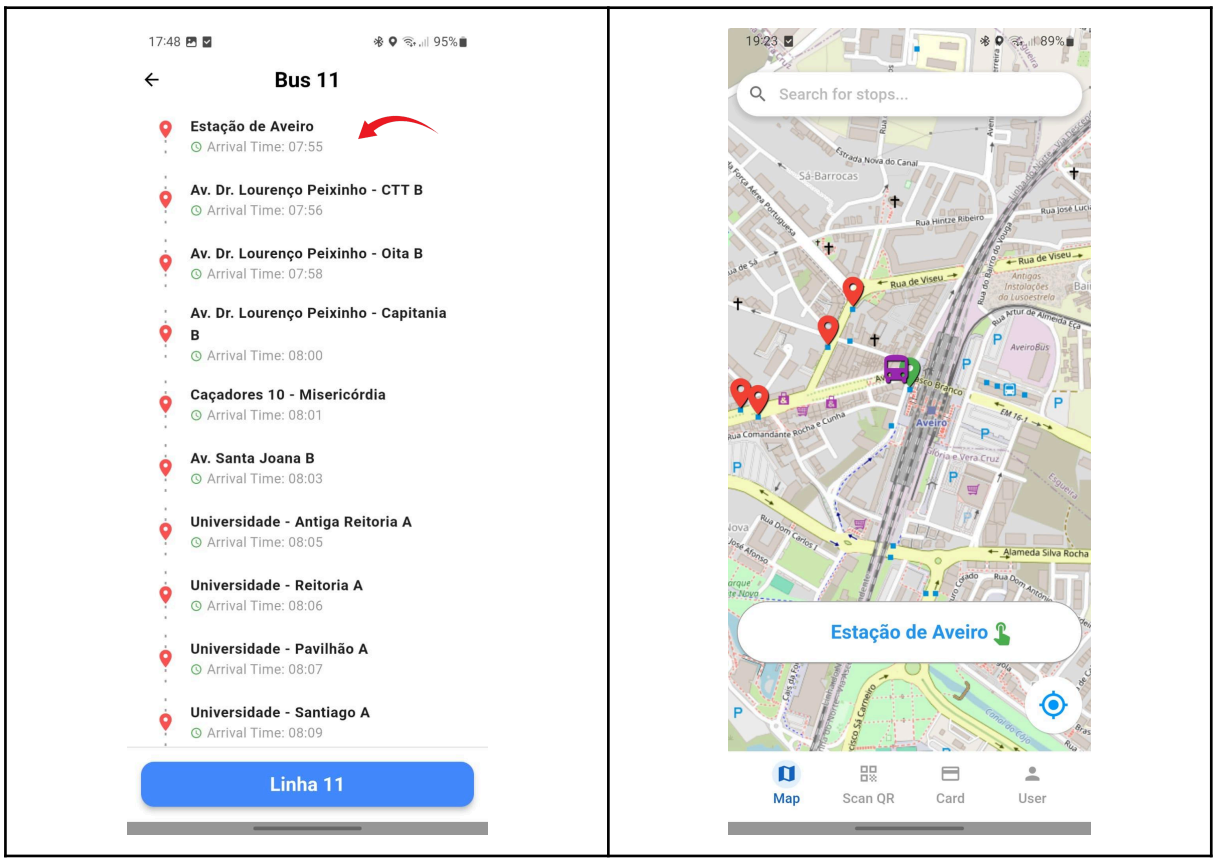


Figure 7 - Selecting bus to see the details (Bus Page), and then selecting a stop redirecting to map page

Now having a list with all buses to pass through a stop (Figure 6) you can view the detailed list of stops that each bus will do with the respective arrival times at each one (Figure 7 left). If you are interested in knowing where one of the bus stops is you can click on top of its name and the app will navigate to the map page again showing in green the stop that you've just pressed (Figure 7 right). You can also see all buses of the route that the bus you just selected belongs to by clicking on the bottom button "Linha X" (Figure 7 left) where X is the number of the route.

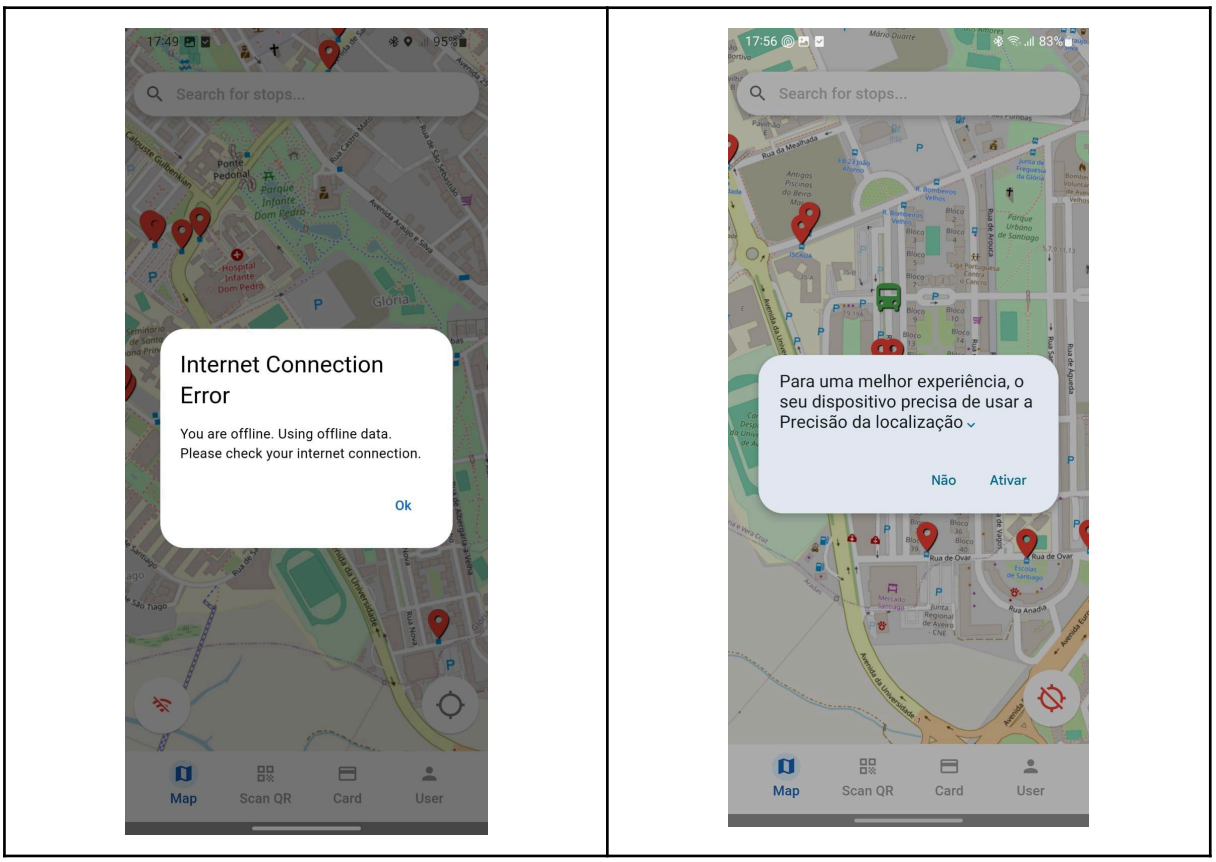


Figure 8 - Connectivity and location awareness on map page

Figure 9 - Scan QR Code page (on the left, showing what the camera is capturing) and QR Code example (on the right)

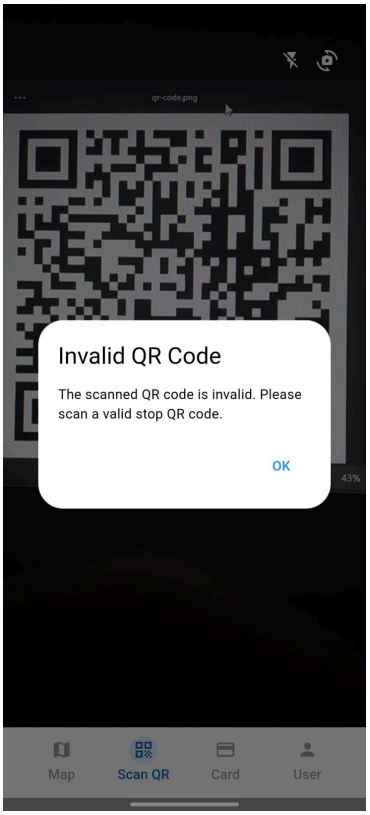


Figure 10 - Reading a wrong QR Code

Imagine that you are next to a stop and you want to know the buses scheduled to stop there, navigating to the Scan QR page (Figure 9 left) and scanning a QR code like the one in Figure 9 right you will get you the same list that we previously talked of in Figure 6.

If the QR Code you just scanned doesn't refer to any stop you will get a message like the Figure 10 one.

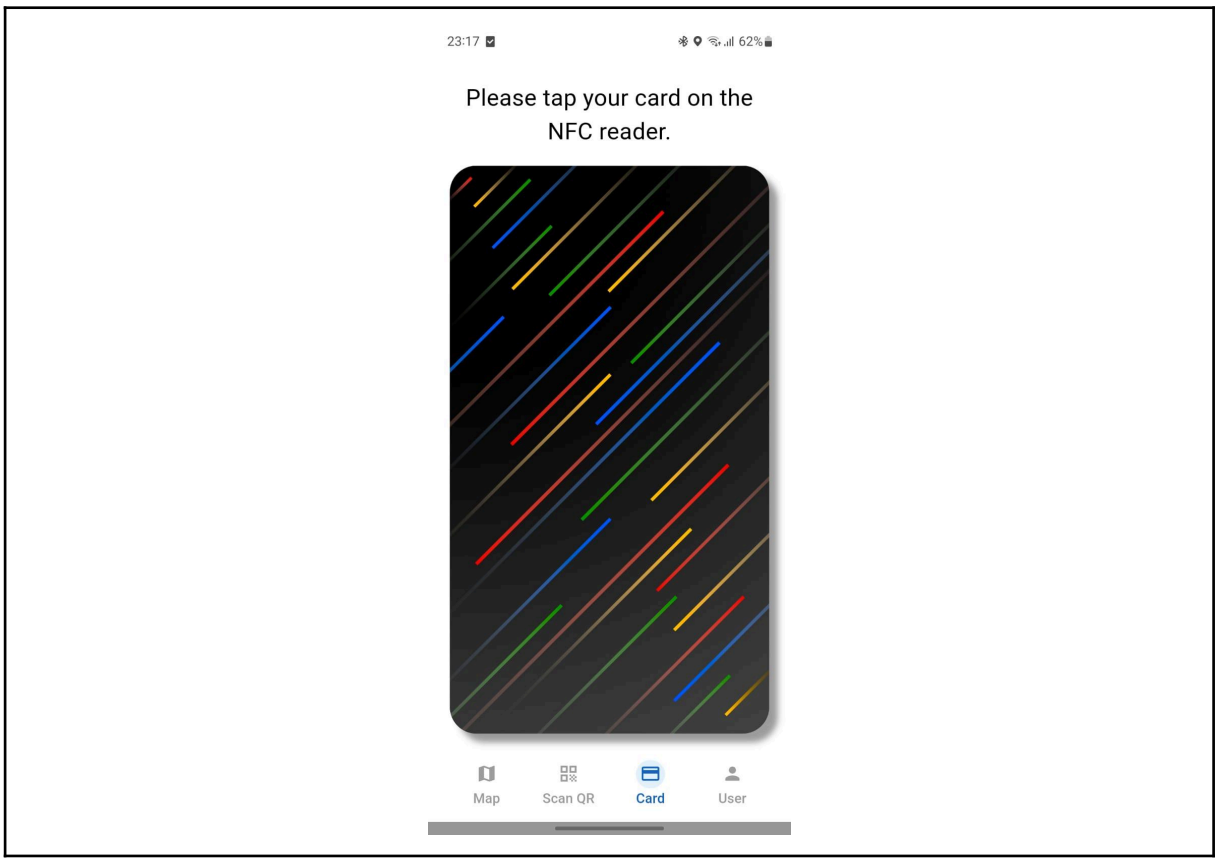


Figure 11 - NFC Card page

When you need to validate your card on a BusTracker Validator you will need to navigate to the Card page (Figure 11) and have your NFC feature enabled. After that you just need to bring your phone NFC closer to the other NFC sensor and the validator will show a screen with the result of your validation.

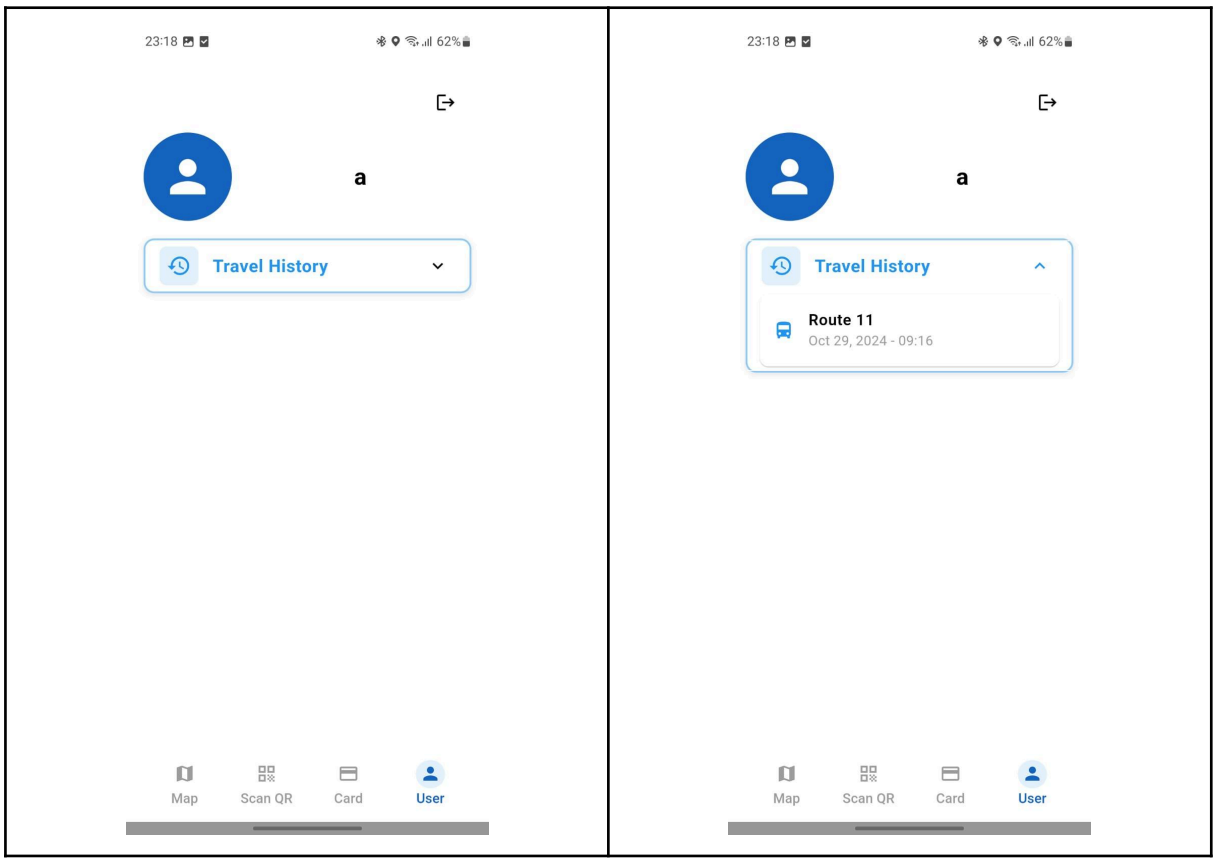


Figure 12 - User page with Travel History closed (on the left) and opened (on the right)

Finally you have your User page where you can Logout (Top-Right Corner Icon) and you can also see your travel history. You just need to press the Travel History envelope (Figure 12 left) and all the travels that you validated with the Card will appear under the envelope (Figure 12 right). Each time you close the envelope a new fetch is done and stored locally so that you can access it offline as well as all the other data in the app.

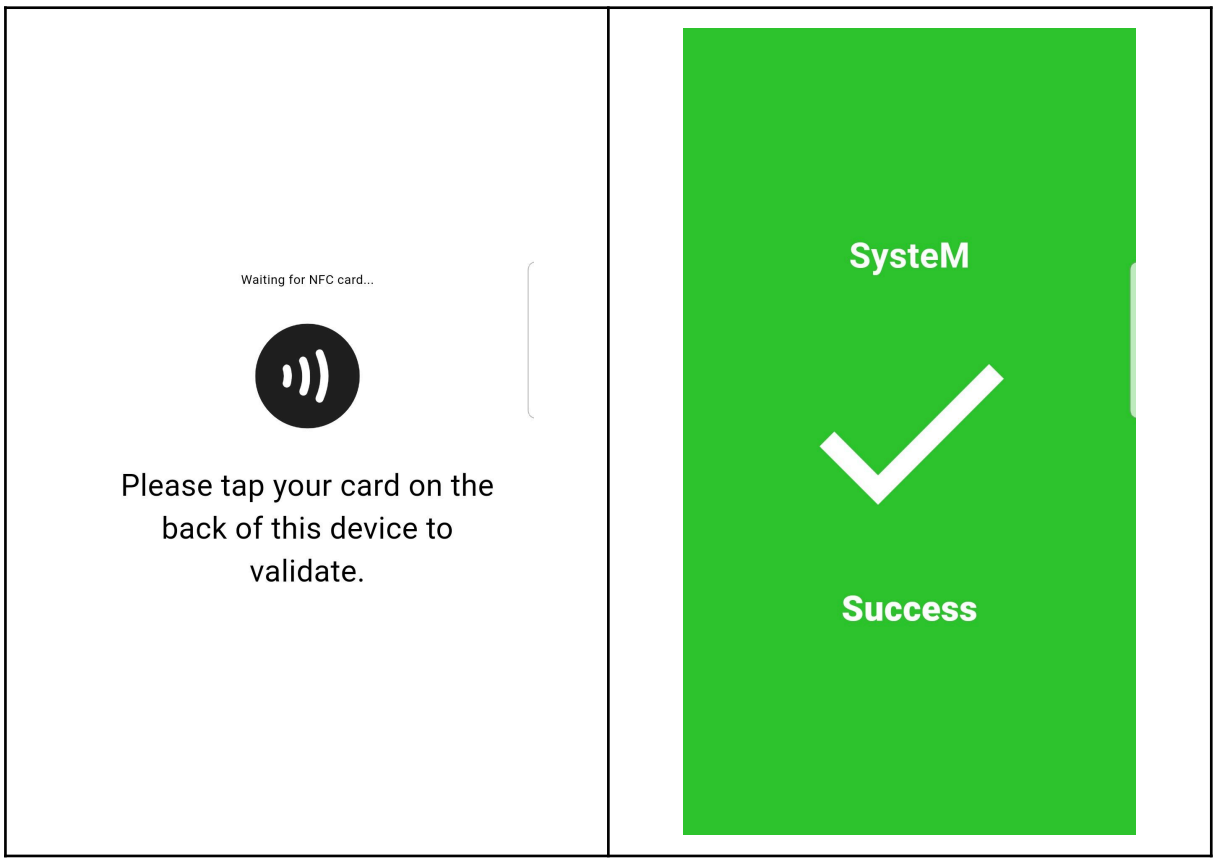


Figure 13 - BusTracker Validator application landing page (on the left) and successful validation screen from BusTracker Validator with user name on top of the screen (on the right)

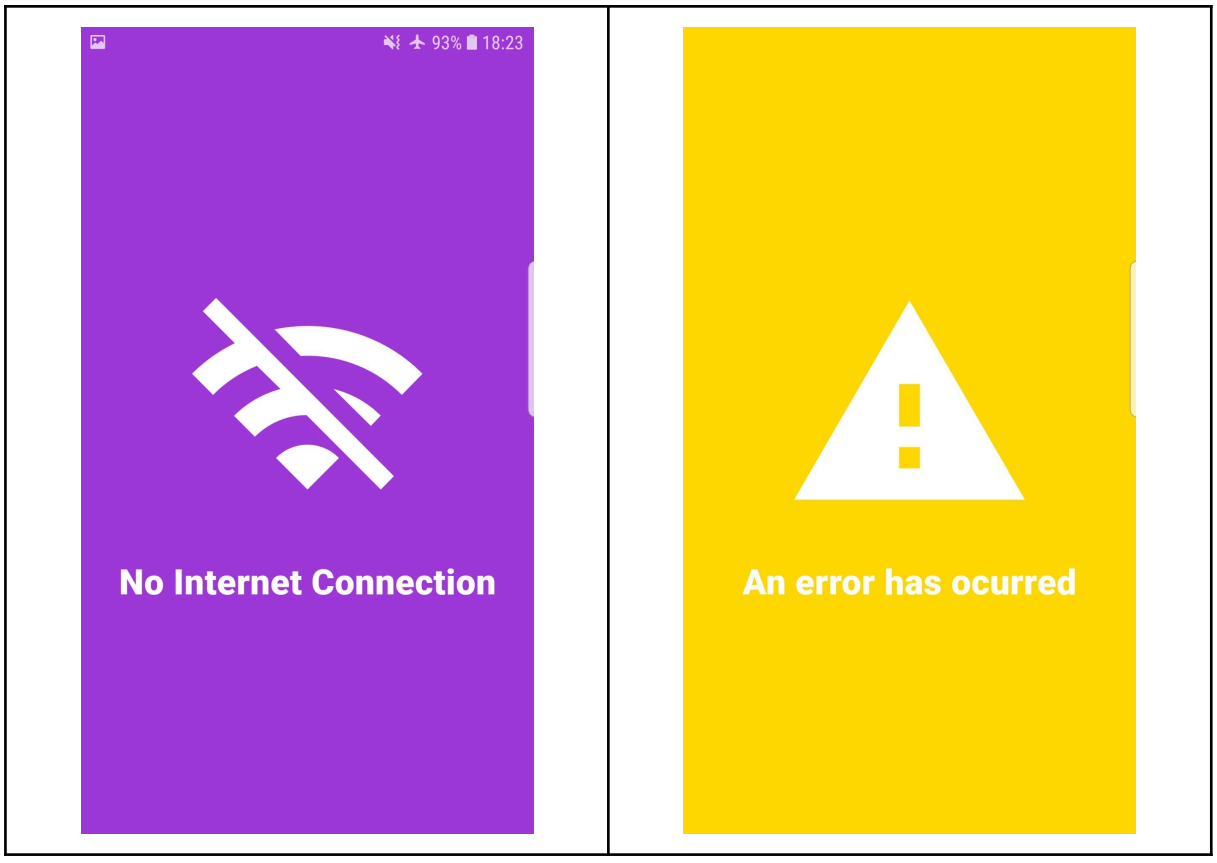


Figure 14 - No connection screen (on the left) and unknown error screen (on the right) both from BusTracker Validator

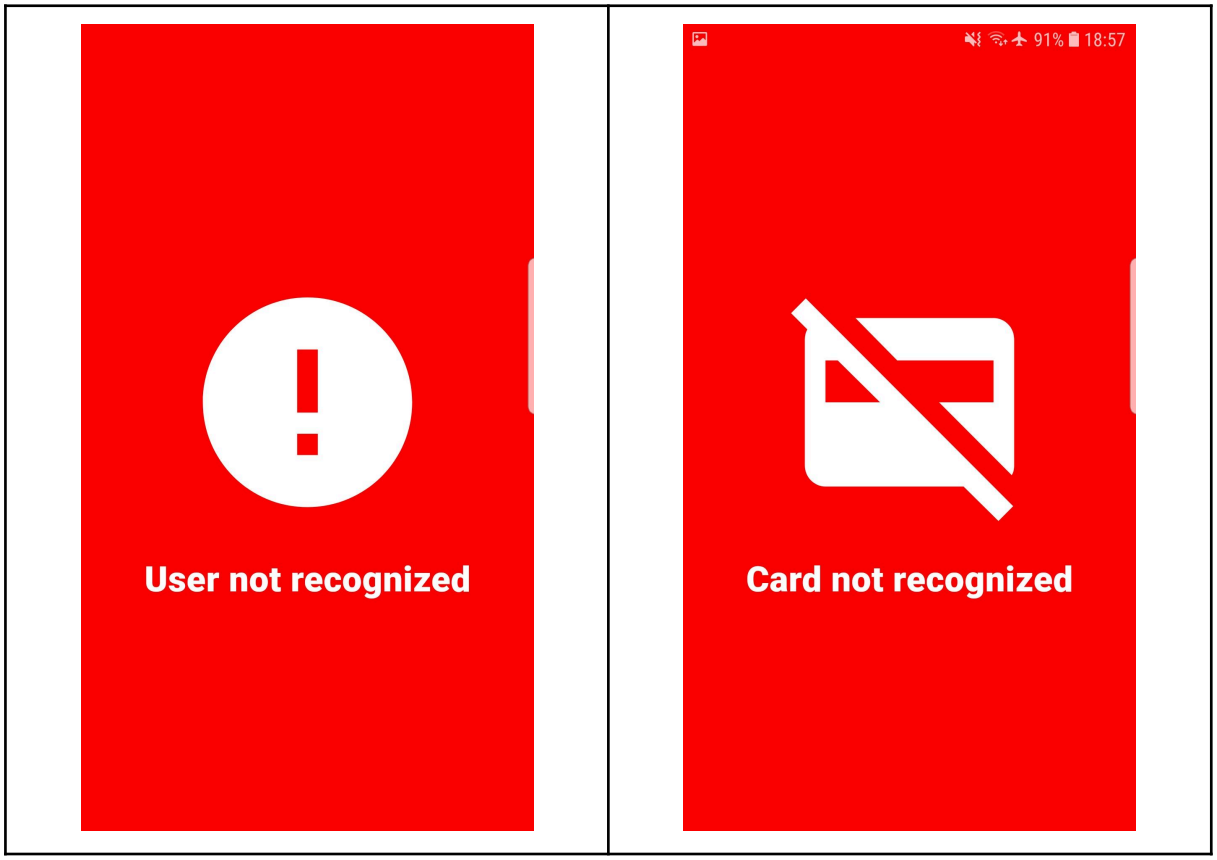


Figure 15 - Unrecognized user screen (on the left) and unrecognized card screen (on the right) both from BusTracker Validator

In the BusTracker Validator app, validating a valid card causes the application to switch between the screens in Figure 13, from the left on to the right one, displaying the user's name at the top of the green screen, simply bringing the devices closer together.

If the app is unable to connect to the internet to update the travel registry, it will display the purple screen shown in Figure 14. In case of an API-level error, the yellow screen from Figure 13 will appear. The screens in Figure 15 will be shown if the card's user is not registered in the system or if the validated card is not recognized by the system.

3 Project Limitations

- Only some information was mined about 2 AveiroBus lines, so there are more timetables for existing lines, stops for other lines and timetables for these other lines.

- Unfortunately, it is not possible for us to monitor the position of real buses on the road in real time, so the buses shown on the map are a simulation carried out with GPX points and an algorithm that tries to approximate the real duration of the trip.

4 Conclusions

In conclusion, we successfully developed all the features we initially planned. We aimed to keep the design straightforward while ensuring everything was accessible and fully functional. This included handling connectivity and location awareness, with the use of Streams, as well as displaying error modals whenever issues arose (Figures 8, 11, 13 and 14).

Lessons Learned:

Initially, managing state updates was straightforward for us, especially for tasks like centering the map on a specific marker on the Map page. Since our widgets were structured in a simple tree, we could achieve this by pushing and popping routes. However, as we introduced new features, particularly the QR code reader for quick access to bus schedules, we encountered a challenge: we needed to redirect users from the QR code reader directly to the map page while passing relevant state information. This became complex because the map was no longer within the same widget tree, making state-sharing less straightforward. To address this, we implemented **Providers** to handle state management. Besides those implementations we also made the connection status available to all pages using a Provider to notify every connection status update. Using Providers allowed us to manage state more effectively across disparate parts of the app, ensuring that the map could center on the correct marker even when accessed via the QR code reader.

Work distribution:

- Guilherme Antunes - 50%

  Main contributions:
  - GPS/Location Services/Map
  - NFC features
  - API & Databases
  - Data mining
- Pedro Rasinhas - 50%

  Main contributions:
  - Camera/QR Codes
  - Authentication features

- User related features
- Navigation & Migration to Providers

5 Supporting Resources

Code repository:	https://github.com/SysteM1922/projeto-cm-flutter
Ready-to-deploy APK:	(Main App) https://github.com/SysteM1922/projeto-cm-flutter/blob/main/projeto_cm_flutter/build/app/outputs/flutter-apk/app-release.apk (Validator App) https://github.com/SysteM1922/projeto-cm-flutter/blob/main/nfc_reader/build/app/outputs/flutter-apk/app-release.apk
API Documentation:	http://4.175.122.244/docs

6 User Manual

The figures in the 2.4 section, along with their descriptions, can be used as a user manual for both applications.