universidade de aveiro
departamento de eletrónica, telecomunicações e informática

CM - 2024/2025 **G02 Project Report**

RouteTrack

Trip Itinerary Planner App

108317 Miguel Miragaia
108287 Miguel Belchior

Contents

1 Introduction

This report represents the first project undertaken as part of the "Mobile Computing" curriculum. The main goal of this project was to design and develop a mobile application that addresses a specific need or problem within the realm of mobile computing. In line with this objective we created RouteTrack, a mobile application designed to facilitate trip itinerary planning and tracking of the user location in said itinerary. RouteTrack is also available as a Wear Os app, as it works as an independent application after receiving the trip data from the smartphone.

In the subsequent sections, we offer an in-depth exploration of Routetrack, the methodology employed to address the challenge, the structural framework, an evaluation of accomplished goals, individual contributions, and a user-oriented guide.

2 Motivation

The development of RouteTrack stemmed from the recognition of a pressing need in the realm of mobile computing: the efficient and intuitive trip itinerary application, with a smartwatch integration.

3 Solution

To ensure a seamless and user-friendly interface, our initial focus was conceptualizing the optimal framework and key elements of the RouteTrack app. We initiated this phase by prototyping in figma the app's design and identifying essential widgets and features crucial for its functionality.

- **Splash Page:** Introduces users to the app with a brief logo display during startup.
- **Login and Registration Page:** Allows users to log in or create a new account with required credentials.

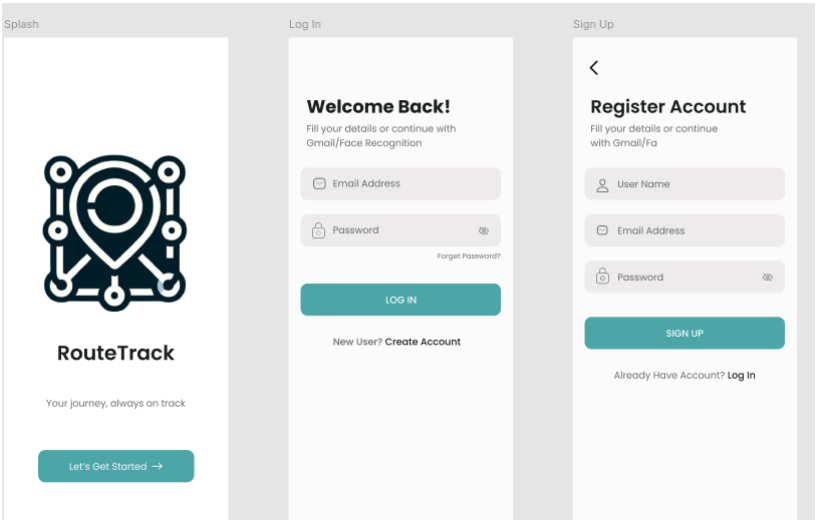

Figure 1: UI/UX Figma Prototype - Splash, Login and Register Pages

- **Home Page:** Serves as the central hub for navigation. Create a Trip or view already created Trips.
- **Create Trip Page:** Enables users to set up a new trip itinerary.

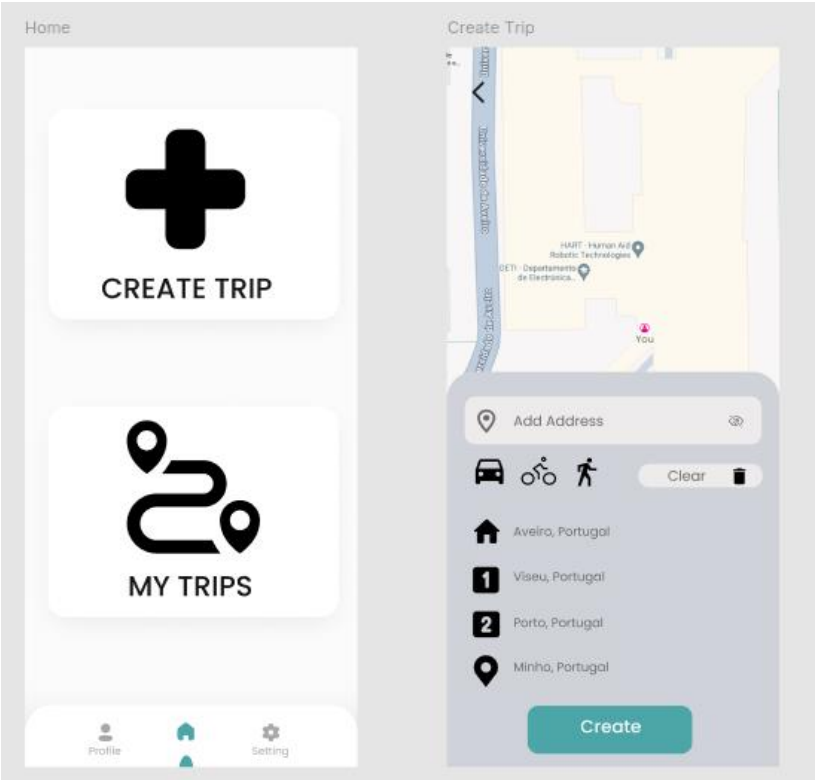

Figure 2: UI/UX Figma Prototype - Home and Create Trip Pages

Apart from the previously identified pages on the prototyping phase of this project the final product contained the following additional pages:

- **Show Trips Page:** After creating a trip, users can view the created trips and share it to the smartwatch.
- **Smartwatch Page:** After receiving the data from the smartphone, the trip details are shown on this page and if closer to a marked point reacts to it with a pop-up.

Therefore, we present bellow the sequence of screens that makes up our final app.

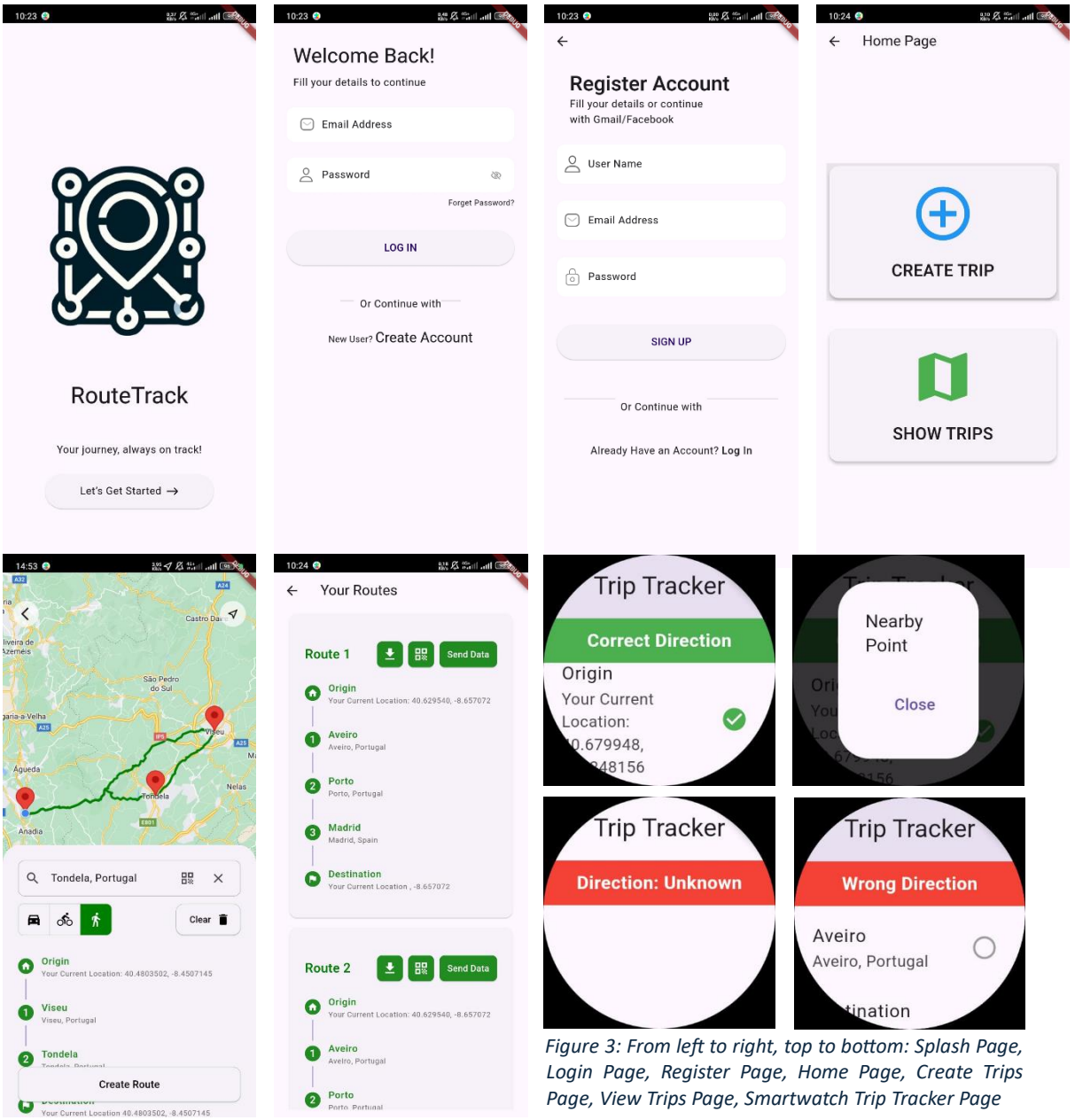

Figure 3: From left to right, top to bottom: Splash Page, Login Page, Register Page, Home Page, Create Trips Page, View Trips Page, Smartwatch Trip Tracker Page

3.1 Features

Smartphone Features prior to the demo

1. **Firebase Login/Registration** – Solution implemented to ensure secure authentication and user management.
2. **Google Maps**:
    - Through google console a project was created to get an API Key to use the google maps map.
    - The google maps widget with markings for each point of the route was implemented with the google_maps_flutter package (using the previously mentioned API Key).
3. **Google Places** - Autocomplete addresses to be added to the route using the google_places_flutter package.
4. **Location/GPS** with geolocator package was used to access the current device's location accurately.
5. **Persistence** was achieved using drift ORM on top of an SQLite relational database.
6. **Permissions** (such as location, notifications and camera) were managed using the permission_handler package.

Smartwatch Features prior to the demo

1. **Sensors-Watch GPS** – Currently it does not exist a package that allows this. However, Kotlin code was used to access the location of the smartwatch.
2. **Smartphone-Smartwatch Communication with MQTT**: The data transfer between the smartwatch and the smartphone was done using Mqtt due to the impossibility and constraints of doing it with Bluetooth.

Smartphone Features subsequent to the demo – as TPC

1. **Google Maps Routes** - google_maps_routes package was used to search the route and draw a polyline between the route points on the google maps map widget.
2. **Generate QrCode** with qr_flutter package to encode a JSON string containing the trip's database values.
3. **Download PDF** – generate a pdf using the pdf package which contains the name of the route as well as the previously mentioned generated QrCode. The package path_provider was also used in order to get the downloads directory;

4. **Notification** – Using flutter_local_notifications package a notification was triggered after the download of said pdf for the route. By pressing the notification the user can open the recently downloaded file (implemented with open_file package).
5. **Scan QrCode with camera** with qr_code_scanner_plus package allowing to receive the previously encoded data and present the shared route to the user so that he can save it if he chooses to.

Smartwatch Features subsequent to the demo – as TPC

1. **Smartwatch Cold/Warm Direction**: Drawing a straight line between the current point and the next point and then drawing a normal between the current location and the closest point on said straight line, calculating the distance we know that if this is greater than the stipulated distance, we know that the user is off the assumed path.

Features tried to implement

1. **Smartwatch Bluetooth data transfer**: We were able to establish a connection between the smartphone and smartwatch. However, there were some challenges related to the Bluetooth data transfer which were not solved due to Wear OS incompatibility reasons.

4 Architecture

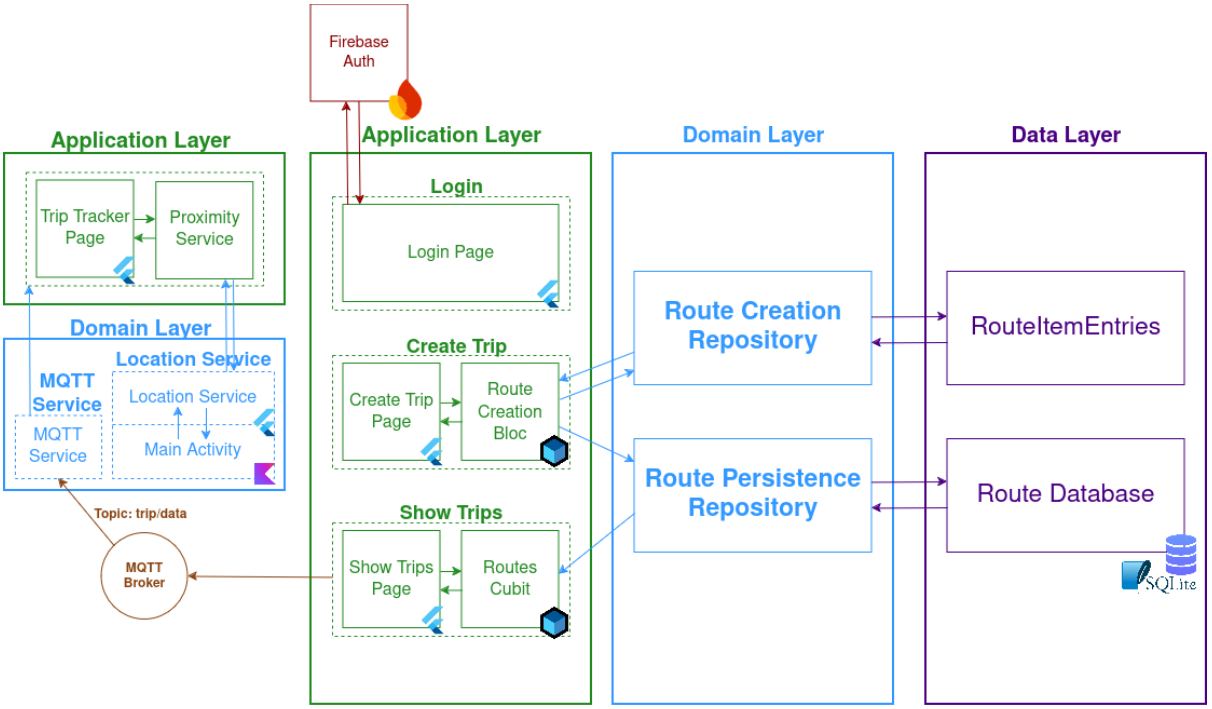

Figure 4: Architecture Diagram

The architecture diagram encompasses two multi-layered systems, including both the smartwatch and smartphone applications.

Smartphone Architecture

The Application Layer includes three different pages (only main pages were included). The **Login Page** communicates with an external firebase authentication service. The **Create Trip Page** and **Show Trips Page** use **Bloc** and **Cubit**, respectively, to separate presentation logic from business logic and data (which in turn in encapsulated and decoupled by using repositories).

Therefore, the **Route Creation Bloc** is responsible for interacting with **Route Creation Repository** to create an in-memory route as the user inputs the addresses for their desired points of interest. The **Route Creation Repository** represents an abstraction for the **RouteItemEntries** linked list which was the chosen data structure to implement the in-memory route as each point of interest gets added before its last element. The **Route Creation Bloc** also flushes this in-memory route to the database by interacting with the **Route Persistence Repository** (which in turn interacts with the **Route Database**). In the **Create Trip Page** a bloc was used to seamlessly handle the different events that could be triggered by the user which include the addition of a route entry, the removal of a route entry, the

clearance of the in-memory route, the scanning and substitution of the route when scanning the QrCode, etc.

For the **Show Trips Page** the **Routes Cubit** was used instead of a bloc due to its simplicity. The sole responsibility of the **Routes Cubit** is to monitor the routes saved on the database so that they can be displayed to the user. To make that possible the **Route Persistence Repository** provides to the **Routes Cubit** a stream of the Route with its points of interest.

Smartwatch Architecture

The initial goal was to communicate with the smartwatch using Bluetooth Low Energy. As that was not possible, the use of a MQTT was adopted as an alternative to sync the data of the trips to the smartwatch.

The **Show Trips Page** sends the data for a trip through the topic trip/date of an **MQTT broker**. The MQTT Service consumes this message and displays the route information in the **Trip Tracker Page**. Within the Application Layer this page interacts with the **Proximity Service** which is responsible for calculating if the user is moving closer or farther from the designated route. In order to do these calculations, the **Proximity Service** needs the current GPS location of the smartwatch which is made available through the **Location Service** which is present on the Domain Layer.

In the **Location Service** as there are not flutter packages to access the smartwatch gps location we used Kotlin code to access it which is represented in the diagram as **MainActivity.kt**. We didn't write this code and acknowledge the author "tgritter" that provided it on the following github issue: https://github.com/Baseflow/flutter-geolocator/issues/1197.

5 Domain Model

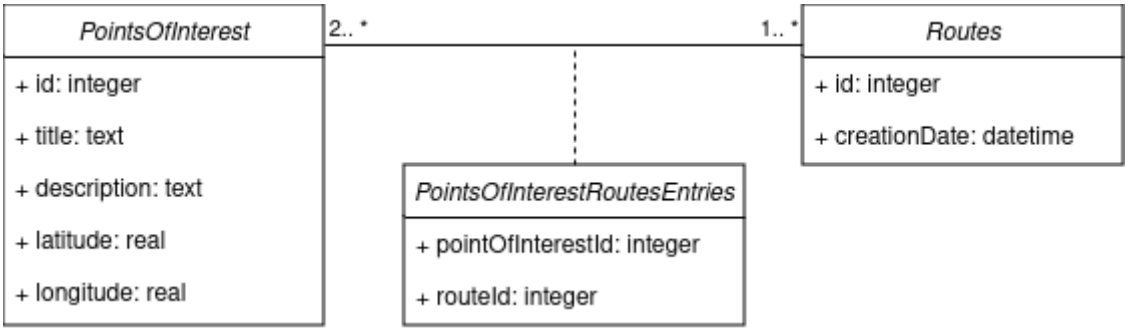

Figure 5: Domain Model UML Diagram

As previously mentioned, the system persists its data on an SQLite relational database. In order to achieve the project requirements, the following database tables were defined:

1. **Routes** – The trip route that passes through various **PointsOfInterest**.
    - **id** – unique identifier of the route.
    - **creationDate** – the date and time the route was created.

2. **PointsOfInterest** – These are the individual locations the user pretends to visit or pass by along their route to their destination.
    - **id** – unique identifier of the point of interest.
    - **title** – name of the point of interest.
    - **description** – brief description providing details about the point of interest, such as the full address.
    - **latitude** – geographic latitude coordinate for the point of interest.
    - **longitude** – geographic longitude coordinate for the point of interest.

3. **PointsOfInterestRouteEntries** – This table represents the many-to-many relationship between **PointsOfInterest** and **Routes** indicating which points of interest are included in each route:
    - **pointOfInterestId** – foreign key linking to the unique identifier of a point of interest. The cardinality of the relationship is “2..*” due to each route having a minimum of two points of interest: origin and destination.
    - **routeId** – foreign key linking to the unique identifier of a route. The cardinality of the relationship is “1..*” due to each point of interest having a minimum of one linked route.

6 Overall Assessment

As a main objective, we aimed to develop a functional app that helps users manage their trip itineraries, provides accurate location tracking and is available to use in a smartwatch. Throughout our assessment, we were able to measure how well RouteTrack achieves these goals by ensuring technical stability across various devices.

6.1 Issues Found

During the development phase, we encountered major problems related to data transfer between smartphone and smartwatch, we used a large diversity of package but none with success. Using flutter_blue_plus we were able to make a connection between the smartphone and smartwatch and write data on a certain characteristic. However, when reading that characteristic there were inconsistencies as the data received was not the one that was sent. The principal characteristic that was supposed to be written was, supposedly, being advertised by the watch but never visible at the smartphone. There was another issue that we faced during the development process, as it does not exist any flutter package to access the Wear Os GPS location, being the solution using Kotlin code to address this issue.

6.2 Contribution Assessment

Miguel Miragaia (50%)
Miguel Belchior (50%)
GitHub Repository - https://github.com/Miragaia/RouteTrack

7. Tutorial

The tutorial of the smartphone app is present in the following link:
CM_G02_108317_108287_RouteTrack_tutorial.mp4

The tutorial/guide for the smartwatch app and its interaction with the smartphone app has the following key points:

1. Press “Send Data” button of the **Show Trips Page**;
2. The route data was sent to the smartwatch which displays the route information (if there isn’t a current route displays direction unknown) and displays a message when the user arrives to a route point;
3. As the user progresses close or farther from the navigation route a correct direction or an incorrect direction message is displayed.

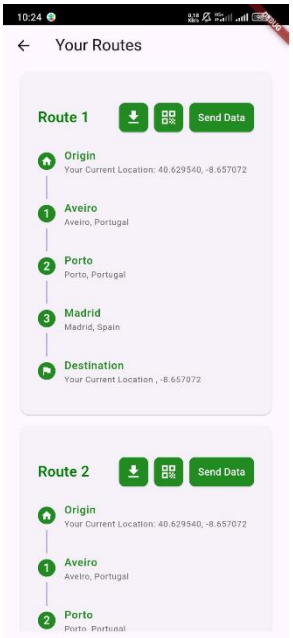

Figure 6: Show Trips Page

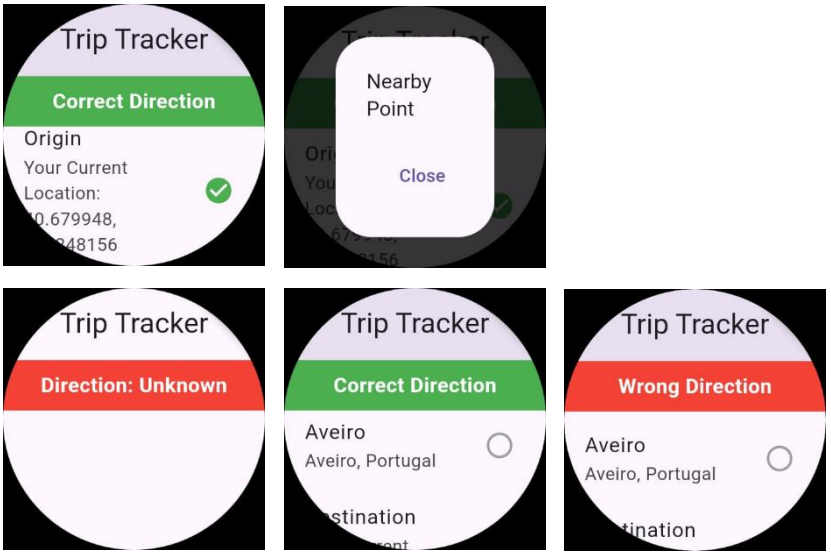

Figure 7: Smartwatch Tutorial Pages

8. Unused Pages of the App Version with Bluetooth

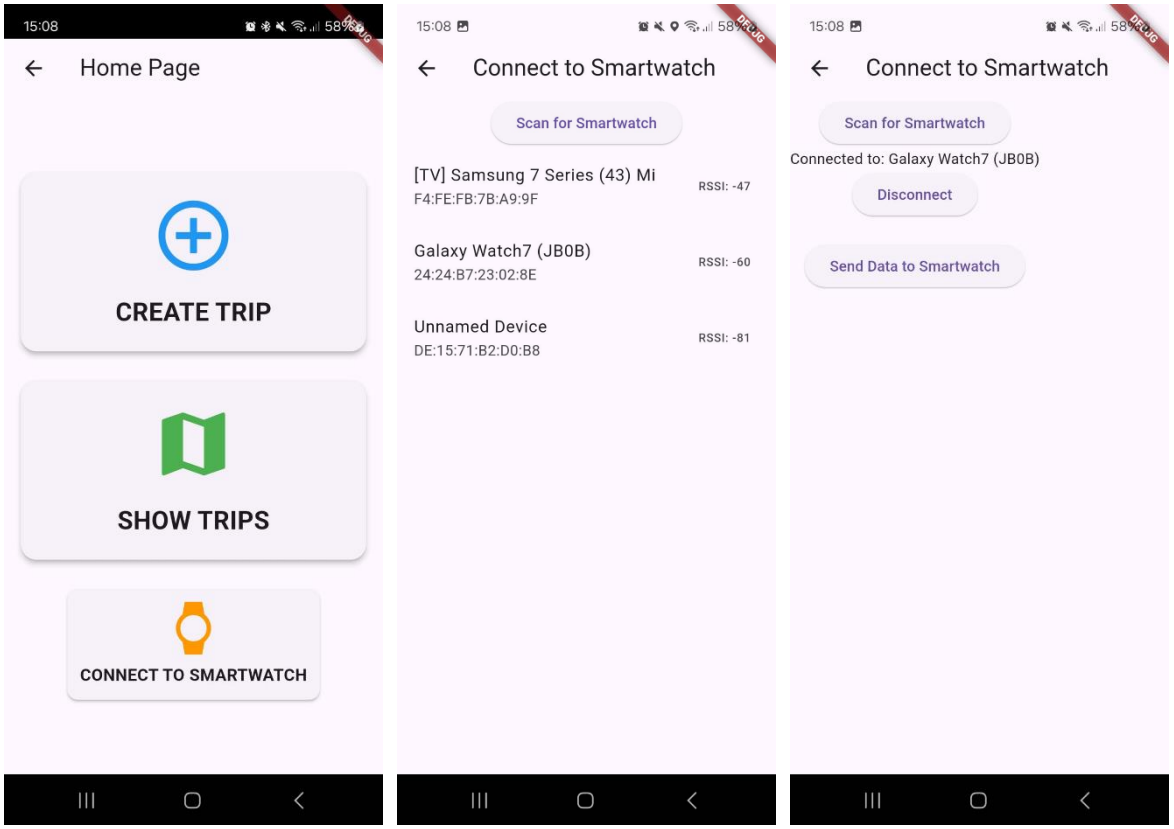

Figure 8: From left to right - Home Page, Device Scanning Page, Send Data to Smartwatch Page