Tech Guide

Getting started

Running the app

To get the app running on your machine:

- Set up Flutter
- Clone the repository (`git clone https://github.com/Eteroclia/Summer_School_2.git`, then `cd Summer_School_2/country_roads`)
- Install dependencies (`flutter pub get`)
- Connect a device or start an emulator.
- Either `flutter run` or start debugging from your code editor.

These steps should allow you to quickly get a developpment build of the app running.
You'll also need to follow the steps in this chapter to get all of the app functionalities working.

Running the DeepFace API

A connection to an instance of the DeepFace API is necessary for face recognition. The code for the API is located in country_roads/api.

To get it running, first create a new venv and download the requirements:
`python -m venv country_roads/api/.venv`
`pip install -r country_roads/api/requirements`

Activate the venv using the corresponding "activate" script in `.venv/Scripts`, then from the root directory of the project, you can then launch the api:
`python ./country_roads/api/src/api.py`

It's important to always start the API from the same directory, as is will use the PWD as the location where to save user pictures and the face-detection database.

Once the server has started, take note of the public address on which the server listens for requests, as you'll need to set it in your .env for your app to connect to it.

Configuration

Credentials and other sensitive data are stored in the .env file.

To set it up on your machine, you'll need to copy the example.env located in `./country_roads` and rename it to `.env`.

You can then configure the different variables present in the .env:

key	description

key	description
API_URL	The address the program will connect to for DeepFace api-related calls. Set it to the public address displayed when launching your local api server (i.e. not the 127.0.0.1 one). When deploying, it should be set to the address of the hosted API.
API_Transport_Key	The key used by the program to connect to the opentransportdata.swiss API, where our travel data comes from. Obtained from a opentransportdata.swiss developper account.

Quick technologies overview

Mobile App Framework: Flutter

Flutter is an open source *framework* by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

Mobile App Programming Language: Dart

Dart is a programming language designed by Lars Bak and Kasper Lund and developed by Google. It can be used to develop web and mobile apps as well as server and desktop applications. Dart is an object-oriented, class-based, garbage-collected language with C-style syntax.

Face recognition: Deepface

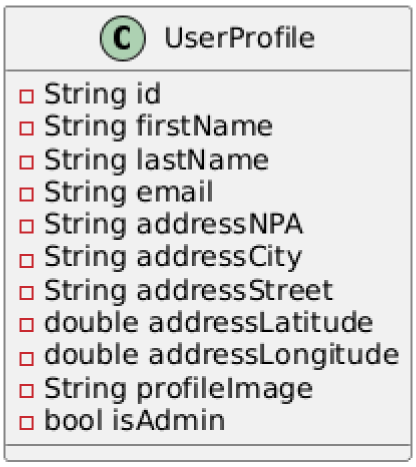
DeepFace API is a powerful facial recognition and analysis tool that provides state-of-the-art capabilities for face detection, facial recognition, emotion analysis, age and gender prediction, and more. It leverages advanced deep learning models to deliver high accuracy and performance across various facial analysis tasks. The API is compatible with multiple deep learning frameworks making it versatile and easy to integrate into existing systems. With support for various backends and customizable options, DeepFace API offers a robust solution for developers looking to incorporate facial analysis features into their applications.

Data Storage: Firestore Database

Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. It’s a NoSQL document-oriented database that stores data in documents, which are organized into collections. Each document contains fields with values of various data types, such as strings, numbers, and binary data.

Models

UserProfile (lib/models/user_profile_model.dart)



The UserProfile class represents the structure of user documents stored in Firestore. It encapsulates various attributes related to a user's profile, including personal information and address details. This class is essential for managing user data within the application, ensuring that all relevant information is organized and easily accessible.

This class mirrors the structure of the "user" documents in Firestore, ensuring seamless integration and data consistency between the application and the database.

The `id` is set to the corresponding Firebase Authentification user's UID during account creation.

External APIs and libraries

Transport

We use opentransportdata.swiss for train journey data.

opentransportdata.swiss allows us to request itineraries between two locations identified by lat-long coordinates.

Weather

We use api.open-meteo.com for our weather data-related needs.

It allows us to retrieve the weather and temperature forecast at a certain lat-long position.

Geocoding

We use the geocoding package from *baseflow.com* as our geocoding solution.
It allows us to get lat-long coordinates from an user-input address.

DeepFace API

For our face-recognition needs, we leverage DeepFace's Python library.

We serve the few functions we need through our own API, as some sort of thin wrapper, to allow us to use it from the app.

Routes

/upload_pic

- **Method** POST
- **Description** Uploads a base64 image and saves it on the server to the `./pics` directory
- **Request Body (json)**
  - `image_data` (required): b64-encoded image data
  - `user_id` (required): Firebase auth user identifier
- **Response**
  - `200 OK` Image uploaded with success. `{'result': 'ok'}` sent as response body.
  - `400 Bad Request` Missing `image_data` or `user_id`. Error message in `error`

/upload_pic is called whenever the user sets a profile picture, either during account creation or when uploading a new picture from their user profile page.
The picture is sent to the API that stores it to the './pics' directory. Only the latest picture of an user is saved.

/find

- **Method** POST
- **Description** Tries to find the Firebase auth user identifier for the person in the uploaded picture
- **Request Body (json)**
  - `image_data` (required): b64-encoded image data
- **Response**
  - `200 OK`
    - If `result` is ok in response body, then user id is returned in `user_id`
    - Otherwise, the API didn't find any match.
  - `400 Bad Request` Missing `image_data`

/find uses DeepFace's `DeepFace.find` function to find the user in the picture.
`DeepFace.find` automatically computes the pictures embeddings, and caches them to a `.pkl` file. This process takes some time, and is run each time new picture files are found.
As this function requires a file, we temporarly save the uploaded image to `./temp_pics`.
Once the whole process is finished, we delete the picture from `./temp_pics`.

/count_faces

- **Method** POST
- **Description** Counts the number of faces in the uploaded base64-encoded image.
- **Request Body (json)**
  - `image_data` (required): b64-encoded image data
- **Response**
  - `200 OK` Found number of faces is returned in `count` in the user body. If no faces were found, `0` is returned.
  - `400 Bad Request` Missing `image_data`

/count_faces uses DeepFace's `DeepFace.extract_faces` function to count faces in the picture.
As this function requires a file, we temporarly save the uploaded image to `./temp_pics`.
Once the whole process is finished, we delete the picture from `./temp_pics`.

Project structure

country_roads/lib

Main folder of for the Flutter project source.

Directories and Files:

- **models/**: Contains data models used in the application.
- **providers/**: Contains provider classes for state management.
- **services/**: Contains service classes for handling business logic and communication with external APIs or databases.
- **views/**: Contains the UI screens of the application.
- **widgets/**: Contains reusable UI components.
- **main.dart**: The entry point of the Flutter application.

Services

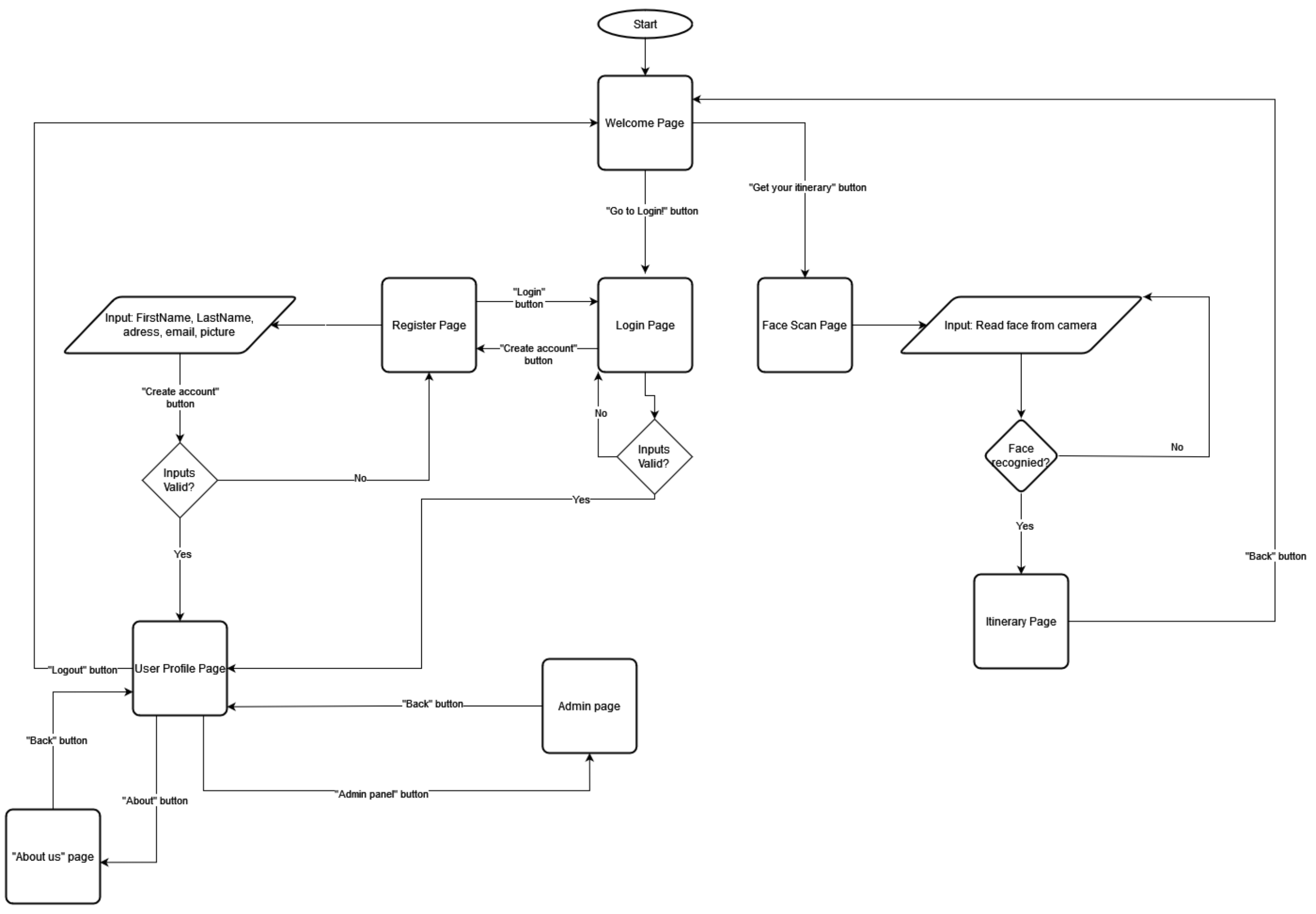
- **AuthService**: Manages user authentification by communicating with Firebase's Auth system.
- **DeepfaceApiService**: Handles queries to the DeepFace API.
- **GeolocService**: Serves positional data through the device's GPS.
- **UserProfileService**: Manages user data by communicating with Firestore Database.
- **TransportDataService**: Retrieves data related to public transport journey through `opentransportdata.swiss`.
- **WeatherService**: Retrieves weather data from `api.Open-Meteo.com`.

country_roads/api

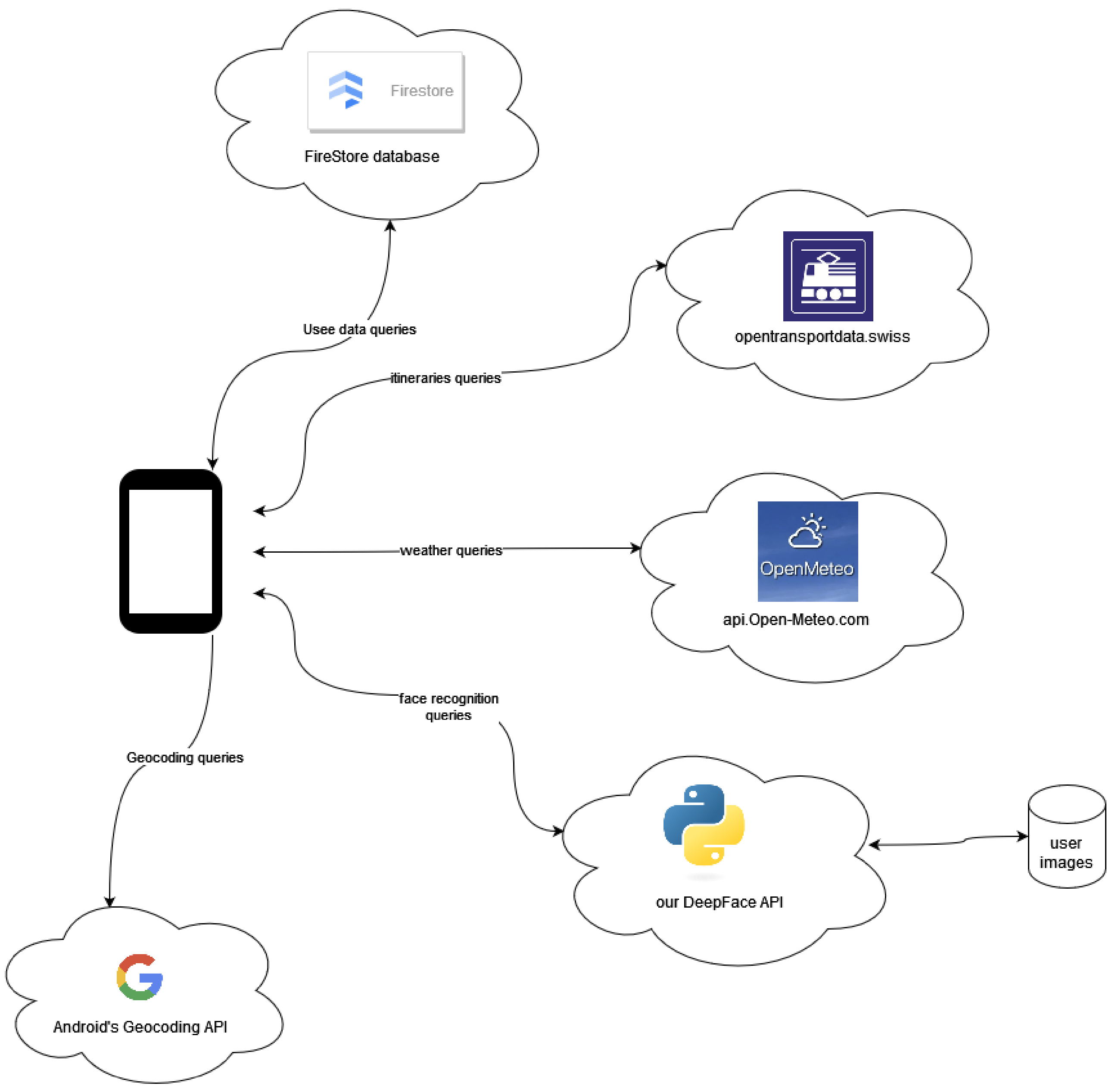
Main folder for our Python DeepFace API

- **src/modules/core/routes.py**: Contains the routes that the DeepFace serves.

UX flowchart



APIs/Web Services Integration Diagram


Firestore
FireStore database
Usee data queries
itineraries queries
opentransportdata.swiss
weather queries
OpenMeteo
api.Open-Meteo.com
face recognition queries
our DeepFace API
user images
Geocoding queries
Android's Geocoding API