# INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

## INSTITUTO POLITÉCNICO DE COIMBRA

Arquiteturas Móveis
LocateWiki – TP2

**Ângelo Galvão Nº 2019138402**
**João Duarte Nº 2020122715**
**Mateus Oliveira Nº 2021136689**

**COIMBRA**

28 de dezembro de 2023

# Índice

# Introdução

Foi proposto desenvolver uma aplicação, em **flutter**. A aplicação a desenvolver tem como objetivo facilitar as visitas turísticas a diversas localizações, disponibilizando informação sobre os locais de interesse existentes nesses locais.

## Tecnologias utilizadas

A linguagem utilizada para o desenvolvimento da aplicação foi o **dart** através da Framework **flutter**, tendo sido usadas apenas as bibliotecas nativas do flutter bem como **shared_preferences** e **locations** (uso solicitado no enunciado). A aplicação recorre à informação criada no âmbito do trabalho de *Kotlin*.

# Ligação à base de dados

Em relação a base de dados foi usada a mesma estratégia do trabalho em kotlin, ou seja, existe um singleton FirebaseHelper responsável por obter e atualizar a informação na base dados.

```
class FirebaseHelper
{
  FirebaseHelper._privateConstructor();

  static final FirebaseHelper _instance = FirebaseHelper._privateConstructor();
  factory FirebaseHelper() => _instance;

  FirebaseFirestore get _firestore {…

  Reference get _storageRef {…

  Future<Set<Location>> getLocations() async…

  Future<Set<Landmark>> getLandmarks(String? locationId) async…

  Future<Set<Category>> getCategories() async…

  StreamSubscription observeLocations(Function(Set<Location>) callback)…

  StreamSubscription observeLandmarks(String? locationId, Function(Set<Landmark>) callback)…

  StreamSubscription observeCategories(Function(Set<Category>) callback)…

  void updateLocationLike(String id, int numLikes) async…

  void updateLocationDislike(String id, int numDislikes) async…

  void updateLandmarkLike(String locationId, String id, int numLikes) async…

  void updateLandmarkDislike(String locationId, String id, int numDislikes) async…

  Future<Uint8List?> getLandmarkImage(String locationId, String landmarkId) async {…
}
```

# Funcionalidades

| Funcionalidade | Implementado |
|---|---|
| Pesquisas, filtros, ordenamentos e visualização da informação | Sim |
| Obtenção da localização e a sua localização no contexto da aplicação | Sim |
| Acesso rápido aos últimos 10 locais de interesse consultados | Sim |
| Armazenamento e consulta dos dados a partir de um servidor | Sim |
| Persistência dos dados locais através do shared preferences | Sim |
| Utilização de mapas para mostrar informação georreferenciada | Sim |

# Implementação

No decorrer do desenvolvimento da aplicação sentimos necessidade de utilizar classes auxiliares para facilitar o acesso a informação da base de dados:

## Landmark

Representa um ponto de interesse, contendo toda a informação acerca do mesmo.

```dart
class Landmark
{
  String id;
  String locationId;
  String categoryId;
  String title;
  GeoPoint geoPoint;
  String otherInfo;
  List<String> usersTrust;
  Map<String, int> usersClassification;
  int numLikes;
  int numDislikes;
  String ownerId;

  Landmark({
    required this.id,
    required this.locationId,
    required this.categoryId,
    required this.title,
    required this.geoPoint,
    required this.otherInfo,
    required this.usersTrust,
    required this.usersClassification,
    required this.numLikes,
    required this.numDislikes,
    required this.ownerId,
  });

  factory Landmark.fromMap(Map<String, dynamic> map) {
    return Landmark(
      id: map['id'],
      locationId: map['locationId'],
      categoryId: map['categoryId'],
      title: map['title'],
      geoPoint: map['geoPoint'],
      otherInfo: map['otherInfo'],
      usersTrust: List<String>.from(map['usersTrust']),
      usersClassification: Map<String, int>.from(map['usersClassification']),
      numLikes: map['numLikes'],
      numDislikes: map['numDislikes'],
```

## Location

Representa uma localização e inclui todas as informações acerca da mesma.

```
class Location
{
  String id;
  String title;
  GeoPoint geoPoint;
  String otherInfo;
  List<String> usersTrust;
  int numLikes;
  int numDislikes;
  String ownerId;

  Location({
    required this.id,
    required this.title,
    required this.geoPoint,
    required this.otherInfo,
    required this.usersTrust,
    required this.numLikes,
    required this.numDislikes,
    required this.ownerId,
  });

  factory Location.fromMap(Map<String, dynamic> map) {
    return Location(
      id: map['id'],
      title: map['title'],
      geoPoint: map['geoPoint'],
      otherInfo: map['otherInfo'],
      usersTrust: List<String>.from(map['usersTrust']),
      numLikes: map['numLikes'],
      numDislikes: map['numDislikes'],
      ownerId: map['ownerId'],
    );
  }
}
```

## Category

Representa uma categoria, contém toda a informação sobre a categoria.

```
class Category {
  String id;
  String name;
  CategoryIcon icon;
  List<String> usersTrust;
  String ownerId;

  Category({
    required this.id,
    required this.name,
    required this.icon,
    required this.usersTrust,
    required this.ownerId,
  });

  factory Category.fromMap(Map<String, dynamic> map) {
    return Category(
      id: map['id'],
      name: map['name'],
      icon: CategoryIcon.fromString(map['icon']),
      usersTrust: List<String>.from(map['usersTrust']),
      ownerId: map['ownerId'],
    );
  }

  String getPresentationName() {
    return icon.getPresentationName();
  }
}
```

# Widgets

Os widgets são componentes básicos de construção de interfaces de utilizador. No decorrer do projeto, achámos que por uma questão de organização e de facilidade no desenvolvimento dividir o projeto em widgets.

## History

Widget responsável por mostrar o histórico dos últimos locais de interesse visitados. Faz uso do widget landmark_card para mostrar os detalhes de cada ponto de interesse na lista.

## LandmarkCard

Widget responsável por exibir os detalhes de um ponto de interesse no formato "card", inclui informação sobre o título, coordenadas, categoria, ícone da categoria e contagens de gostos e não gostos. É um widget interativo porque permite ao utilizador ver os detalhes do ponto de interesse, visualizar o mapa e gostar ou não gostar do mesmo.

## LandmarkDetails

Proporciona uma visão detalhada sobre um ponto de interesse, incluindo informações essenciais, a imagem e o mapa.

## Landmarks

Responsável por mostrar os pontos de interesse com opções de filtragem e pesquisa.

## LikeInfo

Responsável por exibir as informações relativas aos gostos e não gostos, incluindo a contagem dos mesmos e os botões para interagir.

## LocationCard

Exibe detalhes de uma localização e inclui informações como título, coordenadas e contagem de gostos e não gostos. É interativo e permite ao utilizador visualizar detalhes sobre a localização, bem como os pontos de interesse associados e interagir com gostos e não gostos.

## Locations

Exibe a lista de localizações, é responsável por fornecer uma interface interativa para explorar localizações, permitindo a filtragem e acesso rápido ao histórico.

## Map

É responsável por exibir um mapa centrado numa determinada localização ou ponto de interesse.

## SearchBar

Fornece uma barra de pesquisa e opções de ordenação para a lista de localizações.

# Location

A obtenção da localização do utilizador é feita através do singleton LocationHelper. O mesmo trata de permissões de acesso a mesma e disponibiliza métodos para registar listeners as suas atualizações. A localização padrão é o ISEC.

```
class LocationHelper
{
  LocationHelper._privateConstructor();

  static final LocationHelper _instance = LocationHelper._privateConstructor();
  factory LocationHelper() => _instance;

  Location get _location => Location();

  LocationData _locationData = LocationData.fromMap({…

  bool _serviceEnabled = false;
  PermissionStatus _permissionGranted = PermissionStatus.denied;

  Future<void> getLocation(Function(LocationData) onLocationFound) async {…

  bool get isServiceEnabled => _serviceEnabled;
  PermissionStatus get permissionStatus => _permissionGranted;

  StreamSubscription<LocationData>? _locationSubscription;

  void startLocationUpdates(Function(LocationData) onLocationChanged) {…

  void stopLocationUpdates() {…

  double calculateHaversineDistance(double lat1, double lon1, double lat2, double lon2) {…
}
```

# Shared Preferences

De forma a conseguirmos guardar dados localmente entre as execuções da aplicação é utilizada a biblioteca *shared preferences.* Esta permite armazenar informação de modo a gerenciar as classificações atribuídas de forma local e a guardar um histórico dos últimos 10 locais de interesse visitados.

## Guardar dados

```
Future<void> _setSharedPreference(String key, String value) async {
  final prefs = await SharedPreferences.getInstance();
  prefs.setString(key, value);
}
```

## Ler dados

```
Future<String?> _getSharedPreference(String key) async {
  final prefs = await SharedPreferences.getInstance();
  return prefs.getString(key);
}
```