

Mapas



Plugin google_maps_flutter

- O primeiro passo é instalar o plugin. As diretivas básicas de instalação se encontram em:

https://pub.dev/packages/google_maps_flutter/install

- Depois é necessário fazer uma série de configurações que se encontram em:

https://pub.dev/packages/google_maps_flutter#-readme-tab-



Obtendo uma chave de API

- A primeira etapa da configuração é a criação de uma chave de API. Isso é feito no seguinte link:

<https://cloud.google.com/maps-platform/>

- A seguir devemos clicar em Console:



- O passo seguinte é criar seu projeto:



Obtendo uma chave de API

- Clicando em API no Console devemos procurar por Maps SDK for Android e Maps SDK for iOS.
- Clicando em Maps SDK for Android podemos habilitar esse serviço.
- Clicando em Maps SDK for iOS podemos habilitar esse serviço.
- Agora procure por **Credenciais** e **Criar Credenciais**.

The image shows three screenshots from the Google Cloud Platform console. The top screenshot shows the 'Credenciais' (Credentials) page with a red arrow pointing to the '+ CRIAR CREDENCIAIS' (Create Credentials) button. The middle screenshot shows the 'Maps SDK for Android' page with the 'GERENCIAR' (Manage) button and a green checkmark indicating the API is active. The right screenshot shows the 'APIs e serviços' (APIs & Services) page with the 'Credenciais' (Credentials) link highlighted in the left sidebar.



Obtendo uma chave de API

- Após clicar em Criar Credenciais, devemos escolher a opção Chave de API.

+ CRIAR CREDENCIAIS


EXCLUIR

Chave de API

Identifica seu projeto usando uma chave de API simples para verificar cota e acesso

- Uma vez criada a chave, devemos restringir seu acesso (por questões de segurança). Deveremos criar duas chaves, uma para Android e uma para iOS.

Chaves de API

<input type="checkbox"/>	Nome	Data da criação ↓	Restrições	Chave	Uso com todos os serviços (últimos 30 dias) ?	
<input type="checkbox"/>	✅ MapsiOS	12 de mai. de 2020	Apps para iOS, 1 API		22	 
<input type="checkbox"/>	✅ MapsAndroid	12 de mai. de 2020	Maps SDK for Android		56	 



Chave Android

- Para trabalhar com a API de Mapas com mapas precisaremos de uma Google API Key.
- Para tanto precisaremos gerar o *fingerprint* SHA1. Isso é feito através da ferramenta keytool instalada junto com o JDK.
- No terminal deve ser usado o seguinte comando:

Linux e MacOS: `keytool -list -v -keystore ~/.android/debug.keystore \`
`-alias androiddebugkey -storepass android -keypass android`

Windows: `keytool -list -v -keystore "C:\Users\<seu_usuario>\.android\debug.keystore" -`
`alias androiddebugkey -storepass android -keypass android`

Em seguida, copie o que apareceu no campo SHA1



Chave Android

- Devemos restringir a chave para Apps para Android.
- Devemos também editar o pacote (usando o pacote da aplicação).
- Por fim, devemos informar a chave SHA-1.

Restrições do aplicativo

Uma restrição de aplicativo controla quais sites, endereços IP ou aplicativos podem usar sua chave de API. Você pode definir apenas uma restrição de aplicativo por chave.

☐ Nenhuma

☐ Referenciadores de HTTP (sites da Web)

☐ Endereços IP (servidores da Web, cron jobs etc.)

☒ Apps para Android

☐ Apps para iOS

Restringir o uso de seus apps para Android

Adicione o nome do pacote e a impressão digital do certificado de assinatura SHA-1 para restringir o uso dos seus aplicativos para Android

Editar item

Nome do pacote *
com.fundos.fundosimobiliarios


Impressão digital para certificação SHA-1 *

CANCELAR CONCLUIR



Chave Android

- Devemos restringir a API para Maps SDK for Android e clicar em **Salvar**.
- Na sequência devemos procurar por API Key (essa será a chave que utilizaremos no nosso App Android)



API Key

Restrições da API

As restrições de API especificam as APIs ativadas que a chave pode chamar

☐ Não restringir a chave
Esta chave pode chamar qualquer API

☒ Restringir chave

1 API

APIs selecionadas:

Maps SDK for Android

Observação: pode levar até cinco minutos para que as configurações sejam aplicadas

SALVAR CANCELAR



Chave iOS

- Na configuração para iOS basta restringir para Apps para iOS e setar o pacote (Product Bundle Identifier).

Nome *

MapsiOS

Restrições de chave

As restrições ajudam a evitar o uso não autorizado e o roubo de cotas. [Saiba mais](#)

Restrições do aplicativo

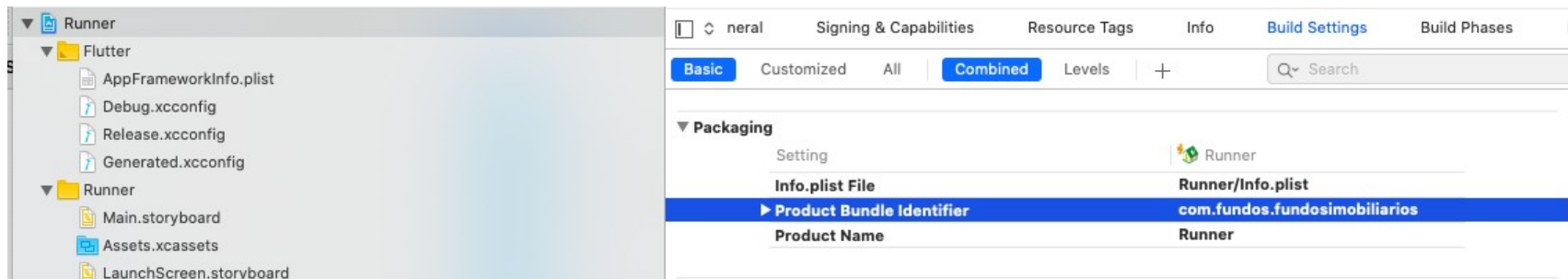
Uma restrição de aplicativo controla quais sites, endereços IP ou aplicativos podem usar sua chave de API. Você pode definir apenas uma restrição de aplicativo por chave.

- ☐ Nenhuma
- ☐ Referenciadores de HTTP (sites da Web)
- ☐ Endereços IP (servidores da Web, cron jobs etc.)
- ☐ Apps para Android
- ☒ Apps para iOS

Aceitar solicitações de um app iOS com um destes identificadores de pacote


com.fundos.fundosimobiliarios

[ADICIONAR UM ITEM](#)



Chave iOS

- Devemos restringir a API para Maps SDK for iOS e clicar em **Salvar**.
- Na sequência devemos procurar por API Key (essa será a chave que utilizaremos no nosso App iOS)



API Key

Restrições da API

As restrições de API especificam as APIs ativadas que a chave pode chamar

☐ Não restringir a chave
Esta chave pode chamar qualquer API

☒ Restringir chave

1 API

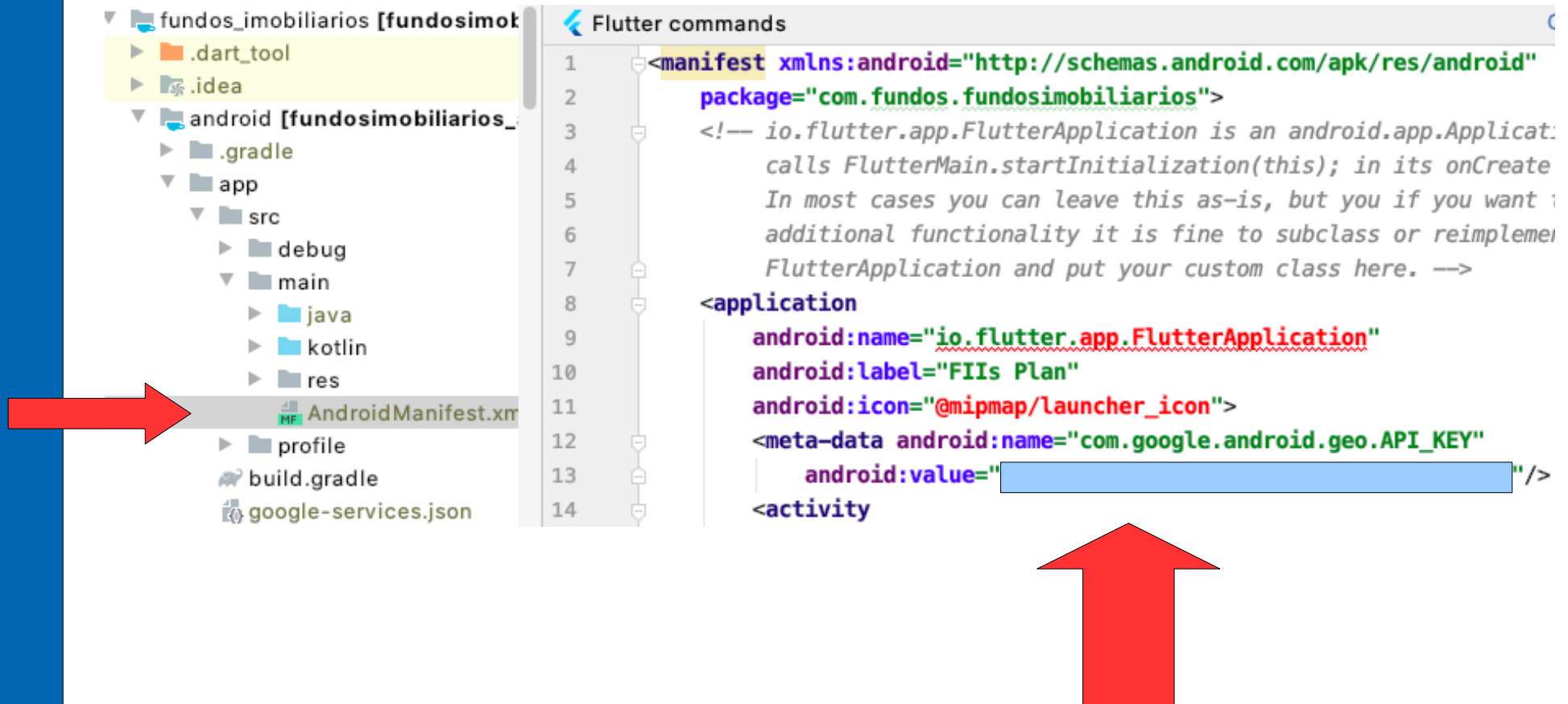
APIs selecionadas:
Maps SDK for iOS

Observação: pode levar até cinco minutos para que as configurações sejam aplicadas

SALVAR CANCELAR



Colocando a API - Android



The image shows a screenshot of an IDE with two panels. The left panel displays the project structure for 'fundos_imobiliarios [fundosimob...]' with a red arrow pointing to the 'AndroidManifest.xml' file. The right panel shows the 'Flutter commands' window with the content of the 'AndroidManifest.xml' file. The code in the right panel is as follows:

```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.fundos.fundosimobiliarios">
3   <!-- io.flutter.app.FlutterApplication is an android.app.Applicat:
4     calls FlutterMain.startInitialization(this); in its onCreate
5     In most cases you can leave this as-is, but you if you want :
6     additional functionality it is fine to subclass or reimplemei
7     FlutterApplication and put your custom class here. -->
8   <application
9     android:name="io.flutter.app.FlutterApplication"
10    android:label="FII's Plan"
11    android:icon="@mipmap/launcher_icon">
12    <meta-data android:name="com.google.android.geo.API_KEY"
13      android:value=" " />
14    <activity
```

A red arrow points to the empty value field for the 'com.google.android.geo.API_KEY' meta-data.



Colocando a API - iOS

The screenshot illustrates the steps to configure the Google Maps API for an iOS Flutter application. It shows the file explorer, the Swift code in AppDelegate.swift, and the Info.plist file.

File Explorer: The 'ios' directory is expanded, showing files like 'AppDelegate.swift', 'GeneratedPluginRegistrant.m', 'GoogleService-Info.plist', and 'Info.plist'. A red arrow points to 'AppDelegate.swift'.

Flutter commands (AppDelegate.swift):

```
1 import UIKit
2 import Flutter
3 import GoogleMaps
4
5
6 @UIApplicationMain
7 @objc class AppDelegate: FlutterAppDelegate {
8     override func application(
9         _ application: UIApplication,
10         didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
11     ) -> Bool {
12         GMServices.provideAPIKey("YOUR_API_KEY")
13         GeneratedPluginRegistrant.register(with: self)
14         return super.application(application, didFinishLaunchingWithOptions: launchOptions)
15     }
16 }
```

A red arrow points to the 'import GoogleMaps' line, and another points to the 'GMServices.provideAPIKey' call.

Info.plist:

```
2 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3 <plist version="1.0">
4 <dict>
5     <key>io.flutter.embedded_views_preview</key>
6     <true/>
```

A red arrow points to the 'io.flutter.embedded_views_preview' key.

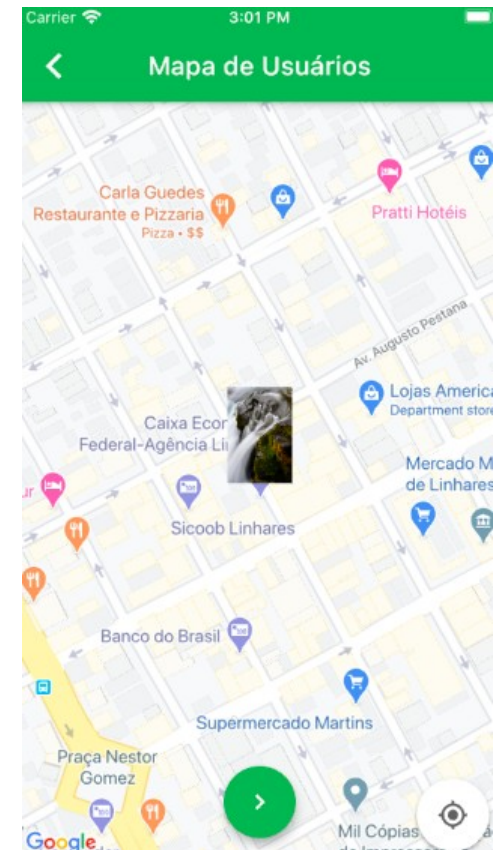
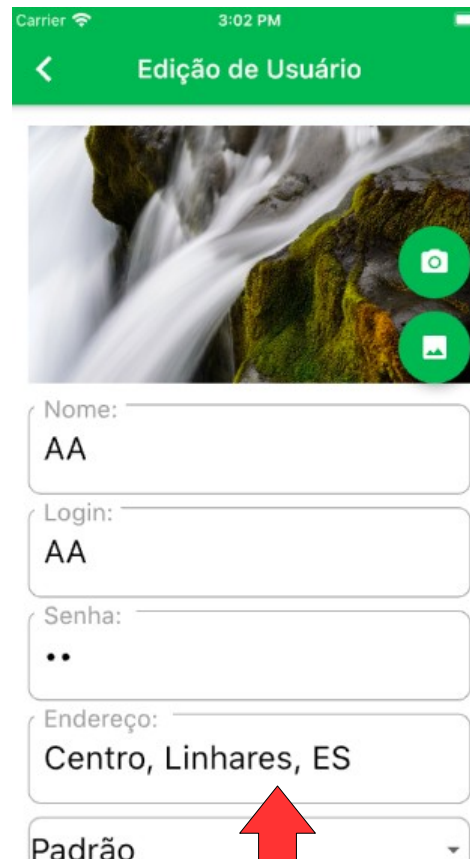
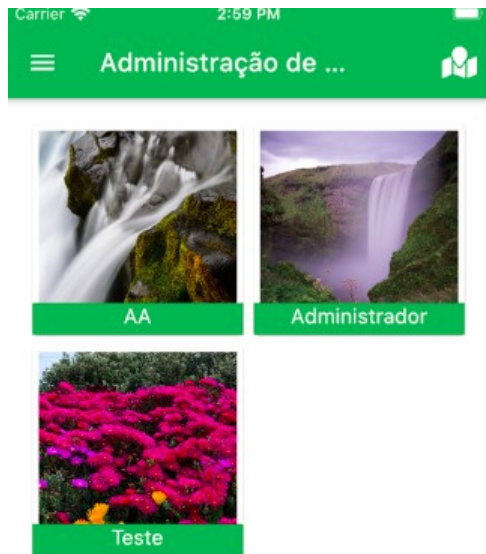


Plugin geocoder

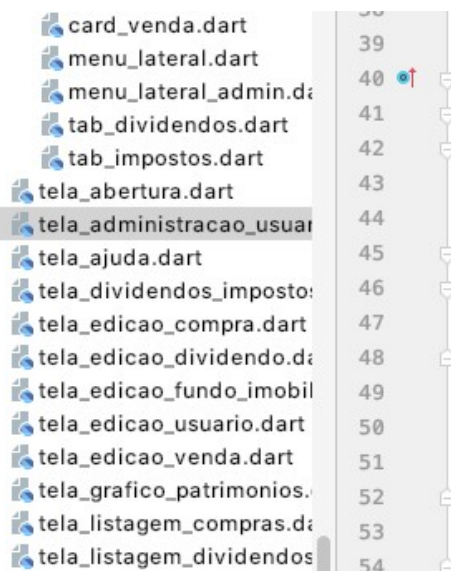
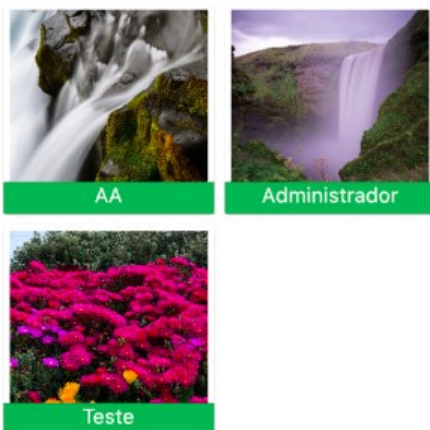
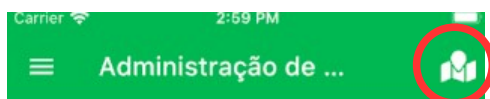
- Além do plugin para trabalhar com mapas precisaremos do plugin para transformar um endereço em coordenadas GPS (latitude e longitude).
- O plugin **geocoder** se encontra no link:
<https://pub.dev/packages/geocoder/install>
- A instalação obedece o mesmo procedimento das que já fizemos até o momento e não há nenhuma configuração adicional que precise se feita para esse plugin ser utilizado.



A aplicação – FIIs Plan



Menu Mapa



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Administração de Usuários"),
      actions: <Widget>[
        IconButton(
          icon: FaIcon(
            FontAwesomeIcons.mapMarked,
          ), // FaIcon
          onPressed: () {
            push(context, TelaMapaUsuarios(_controle.usuarios));
          },
        ), // IconButton
      ], // <Widget>[]
    ), // AppBar
  );
}
```



TelaMapaUsuario

```
class TelaMapaUsuarios extends StatefulWidget {  
  List<Usuario> usuarios;  
  
  TelaMapaUsuarios(this.usuarios);  
  
  @override  
  _TelaMapaUsuariosState createState() => _TelaMapaUsuariosState();  
}  
  
class _TelaMapaUsuariosState extends State<TelaMapaUsuarios> {  
  ControleTelaMapaUsuarios _controle;  
  
  @override  
  void initState() {  
    // TODO: implement initState  
    super.initState();  
    _controle = ControleTelaMapaUsuarios(widget.usuarios);  
  }  
}
```

- Essa tela recebe como parâmetro a lista de usuários a serem plotados no mapa.
- Para tanto, passa essa lista ao controlador que irá apoiar as funcionalidades dessa tela.



TelaMapaUsuario

```
_body() {  
  Future<List<Marker>> future = _controle.obterMarkers(widget.usuarios);  
  return FutureBuilder<List<Marker>>(  
    future: future,  
    builder: (context, snapshot) {  
      if (!snapshot.hasData) {  
        return Center(  
          child: CircularProgressIndicator(),  
        ); // Center  
      }  
      _controle.markers = Set.of(snapshot.data);  
      _controle.inicializarPosicaoAtual();  
      return _conteudo();  
    },  
  ); // FutureBuilder  
}
```

- No método **_body** (corpo do método build) temos a chamada ao método **obterMarkers**. Esse método gera uma lista de markers a partir de uma lista de usuários. Um marker é um marcador colocado no mapa.
- Em seguida transformamos a lista num **Set** de markers (o uso de Set ocorre porque o widget **GoogleMap** trabalha com um Set de markers).
- **_conteudo** irá efetivamente apresentar a informação na tela. Enquanto isso não acontece é apresentado um **CircularProgressIndicator**.



TelaMapaUsuario

- Aqui temos o widget **GoogleMap** com seus markers.
- Foi utilizada uma **Stack** para colocar o botão de avançar nos markers do mapa.
- **_onMapCreate** apenas faz o seguinte código:

```
_controle.mapController = controller;
```

```
_conteudo() {  
  return Stack(  
    children: <Widget>[  
      Container(  
        child: GoogleMap(  
          initialCameraPosition: CameraPosition(  
            target: _controle.obterPosicaoInicial(),  
            zoom: 17,  
          ), // CameraPosition  
          markers: _controle.markers,  
          onMapCreated: _onMapCreated,  
        ), // GoogleMap  
      ), // Container  
      Container(  
        alignment: Alignment.bottomCenter,  
        padding: EdgeInsets.only(bottom: 20),  
        child: FloatingActionButton(  
          child: Icon(Icons.navigate_next),  
          onPressed: () {  
            _controle.avancarProximoMarker();  
          },  
        ), // FloatingActionButton  
      ), // Container  
    ], // <Widget>[]  
  ); // Stack  
}
```



obterMarkers

```
Future<List<Marker>> obterMarkers(List<Usuario> usuarios) async {  
  List<Marker> markers = List<Marker>();  
  for (Usuario usuario in usuarios) {  
    LatLng latLng =  
      await Localizador.obterLatitudeLongitudePorEndereco(usuario.endereco);  
    if (latLng != null) {  
      BitmapDescriptor userIcon =  
        await obterBitmapDescriptor(usuario.urlFoto);  
      Marker marker = Marker(  
        markerId: MarkerId(usuario.id.toString()),  
        position: latLng,  
        icon: userIcon,  
        infoWindow: InfoWindow(  
          title: usuario.nome,  
          snippet: usuario.endereco,  
        ), // InfoWindow  
      ); // Marker  
      markers.add(marker);  
    }  
  }  
  return markers;  
}
```

- Para cada um dos usuários é gerada uma **LatLng** a partir do seu endereço.
- Se **latLng** for nulo significa que o endereço não foi encontrado, ou seja, que não foi possível transformá-lo em coordenadas.
- Na sequência a foto do usuário é convertida num **BitmapDescriptor** (que é o formato definido pelo Marker).
- Cada usuário com endereço válido terá um Marker. Se não tiver foto será usada uma imagem padrão.
- **infoWindow** é uma “janelinha” que mostra os dados **title** e **snippet** quando a imagem é clicada.



```

Future<BitmapDescriptor> obterBitmapDescriptor(String foto) async {
  if (foto == null) {
    return await GeradorBitmapDescriptor.gerarBitMapDescriptorFromAsset(
      'assets/icon/imagem_mapa.png', 100);
  } else {
    File file = await GerenciadoraArquivo.obterImagem(foto);
    return await GeradorBitmapDescriptor.gerarBitMapDescriptorFromFile(
      file, 100);
  }
}

```

- Em obterBitmapDescriptor temos a geração de um BitmapDescriptor a partir do caminho da imagem (parâmetro foto).
- Caso não haja foto é gerado um BitmapDescriptor a partir de uma imagem da pasta **assets**. O segundo parâmetro define o tamanho da imagem (no exemplo 100).
- Caso haja uma foto, o arquivo é obtido e a partir desse arquivo é gerado um BitmapDescriptor.



```

inicializarPosicaoAtual(){
  if(posicao_marker_atual == -1 && markers.length > 0){
    posicao_marker_atual = 0;
  }
}

LatLng obterPosicaoInicial() {
  return posicao_marker_atual == -1
    ? // Centro de Colatina
    ? LatLng(-19.5167339, -40.722392)
    : markers.elementAt(0).position;
}

```

- **inicializarPosicaoAtual** coloca o marcador de posição na primeira posição.
- **obterPosicaoInicial** seta um endereço padrão inicial se não há marker atual. Se existe pelo menos um marker a posição 0 é retornada.



- **avancarProximoMarker** é acionado quando o botão se avançar é clicado. Basicamente ele avança o marcador de posição atual, obtém a **LatLng** dessa posição atual e move a câmera para essa nova coordenada.

```
void avancarProximoMarker() {  
    if(posicao_marker_atual == (markers.length-1)){  
        posicao_marker_atual = 0;  
    } else {  
        posicao_marker_atual++;  
    }  
    final LatLng latlng = markers.elementAt(posicao_marker_atual).position;  
    mapController.moveCamera(  
        CameraUpdate.newCameraPosition(  
            CameraPosition(  
                target: latlng,  
                zoom: 17.0,  
            ),  
        ),  
    );  
}
```



Localizador

```
import 'package:geocoder/geocoder.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';

class Localizador{
  static Future<LatLng> obterLatitudeLongitudePorEndereco(String endereco) async{
    try{
      final query = endereco;

      var addresses = await Geocoder.local.findAddressesFromQuery(query).timeout(new Duration(seconds: 3));
      if(addresses == null) return null;
      var first = addresses.first;
      if (first == null) return null;
      else return LatLng(first.coordinates.latitude, first.coordinates.longitude);
    } catch(erro){
      return null;
    }
  }
}
```



Dúvidas?

