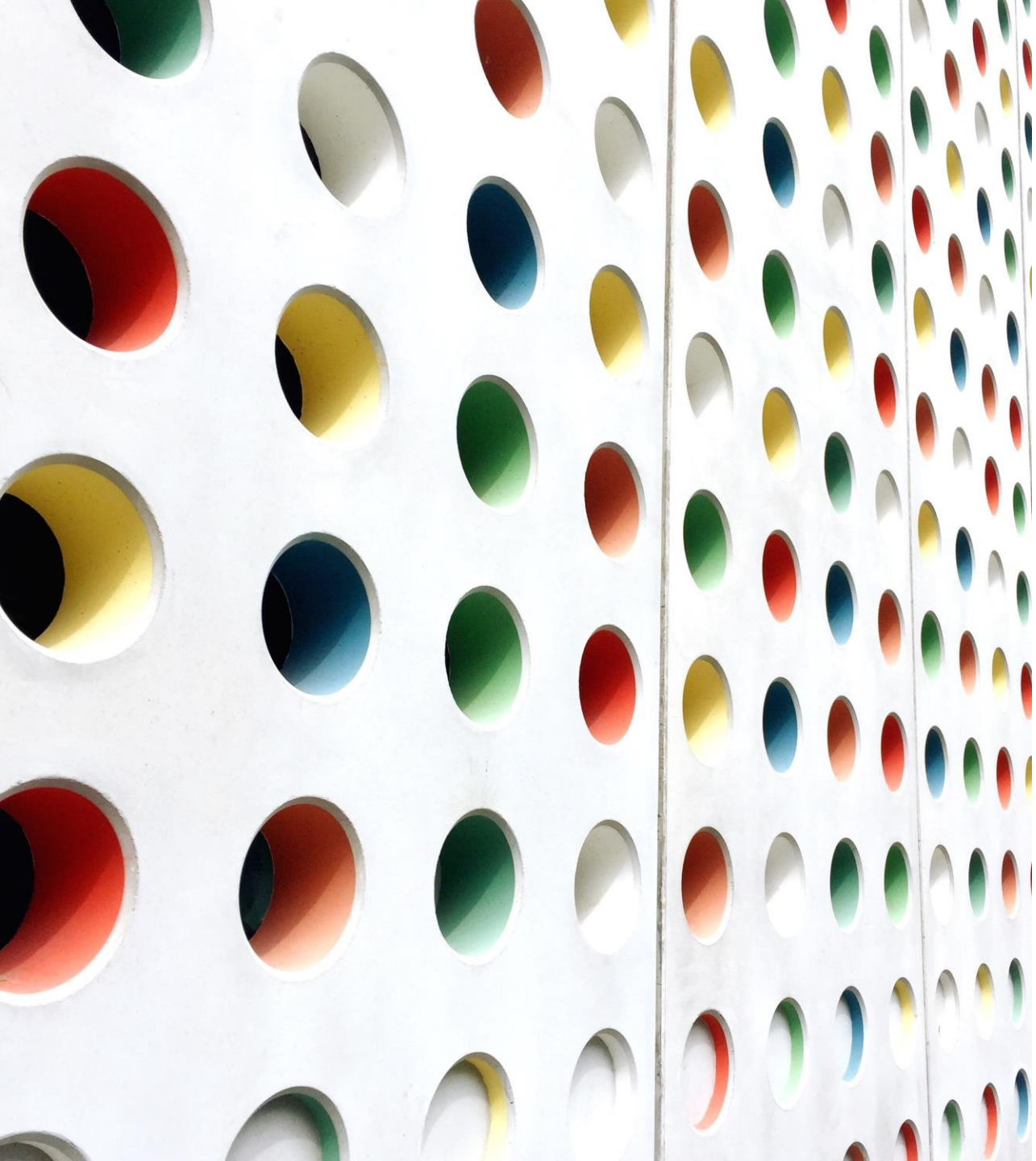


Programação para Dispositivos Móveis

FLUTTER

Professor Jean Carlo Wagner

28/03/2023

Agenda

- *Layout*
- Google Maps: exemplo básico

Layout

- Ajudam a organizar a *interface* de usuário e a estrutura da aplicação de várias maneiras. De certa forma permitem construir o esqueleto do aplicativo.

- O *layout* no Flutter se resume principalmente a uma estrutura de grade, o que implica em linhas e colunas. Assim, há um *widget* Row e um *widget* Column. Cada um deles pode ter um ou mais filhos, e esses filhos são dispostos horizontalmente (ao longo da tela) no caso do *widget* Row e verticalmente (descendo a tela) para um *widget* Column.

```
import "package:flutter/material.dart";

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
```

Layout

```
    return MaterialApp(title : "Flutter Playground",
      home : Scaffold(
        body : Center(
          child : Row(
            children : [
              Text("frase 1 "),
              Text("frase 2 "),
              Text("frase 3 ")
            ]
          )
        )
      )
    );
  }
}
```

Layout

- Inicia-se com a importação da biblioteca material.dart, trazendo o estilo Material *design*.
- A função main() instancia a classe MyApp, passando a instância para a função runApp(). Isso dá ao Flutter o widget de nível superior que ele requer para começar a executar o aplicativo.
- A classe MyApp é um StatelessWidget, já que nesse aplicativo não precisamos de nenhum tipo de estado, e o método build() requerido produz um único *widget* de tipo MaterialApp.
- Esse *widget* implementa grande parte da "estrutura" para nós, logo, é sempre uma boa ideia começar com ele. Como alternativa, podemos usar o *widget* WidgetApp, mas ele demandará a implementação de um código muito mais extenso para a definição pelo menos das rotas (ou seja, telas) da aplicação, portanto, geralmente não é recomendável usá-lo a não ser que tenhamos objetivos específicos a que ele atenda. É bom ressaltar que, mesmo se tivermos desenvolvido para o iOS, poderemos usar MaterialApp como *widget* de nível superior.

Layout

- tittle é uma propriedade de MaterialApp. Neste caso é uma *string* de linha única usada pelo dispositivo para identificar o aplicativo para o usuário. Esse *widget* também pode fornecer outras propriedades, como color, que define a cor primária usada para a aplicação na interface do sistema operacional, e theme, que recebe como valor um *widget* ThemeData e descreve com detalhes as cores empregadas no aplicativo.
- MaterialApp também requer uma propriedade home, e o valor dela deve ser um *widget*, o qual compõe o topo da árvore de *widgets* da tela principal do aplicativo.

- *Widget* Scaffold implementa a estrutura de *layout* básica de uma tela do aplicativo, cuidando de vários elementos comuns da UI, como a barra de navegação de nível superior, os *drawers* (pequenos elementos que deslizam da parte lateral da tela para exibir opções e os *bottom sheets* (como os *drawers*, mas deslizam a partir da parte inferior).
- Outros tipos de *widgets scaffolding* são CupertinoPageScaffold, que é específico do iOS e fornece a estrutura de *layout* básica, incluindo uma barra de navegação superior e conteúdo

Layout

sobre um *background*.

- CupertinoTabScaffold é como CupertinoPageScaffold exceto por incluir uma barra de navegação com abas na parte inferior.

➢ Para usarmos os *widgets* Cupertino, teríamos de adicionar uma importação "package:flutter/Cupertino.dart"; ao aplicativo. Depois, se quiséssemos, poderíamos alterar Scaffold para CupertinoPageScaffold, que então demandaria a alteração de home para child.

➢ Não há restrição para o uso de *widgets* Cupertino em um dispositivo iOS, ou vice-versa. Lembre-se de que o próprio Flutter renderiza a UI em vez de depender do sistema operacional, o que nos permite executar facilmente um tipo de UI na plataforma "errada" se quisermos!

- *Widget* Scaffold fornece várias propriedades, inclusive floatingActionButton, que permite que o aplicativo dê suporte a um *botão de ação flutuante* (FAB, da sigla em inglês).

Layout

- drawer permite que o aplicativo tenha um *drawer* deslizante para funcionalidade oculta.
- bottomNavigationBar permite que o aplicativo tenha uma barra de navegação na parte inferior.
- backgroundColor permite definir a cor de fundo da página.

- *Widget* Scaffold deve ser especificado com a propriedade body.
- O *widget* Center centraliza verticalmente outros *widgets child*, ou seja, por padrão um *widget* Center assume o maior tamanho possível, o que significa que ele preencherá todo o espaço que seu *widget* pai permitir. No exemplo, o *widget* pai é Scaffold, que tem disponível automaticamente o tamanho total da tela, portanto o *widget* Center preencherá a tela inteira.

- *child* de Center é um único *widget* Row, que será centralizado na tela.
- *Widget* Row tem uma propriedade children, que permite especificar um *array* de *widgets*

Layout

que serão dispostos ao longo de Row.

- Um *widget* Text exibe uma *string* de texto em um único estilo. Algumas propriedades interessantes que Text suporta são overflow, que informa ao Flutter o que fazer quando o texto ultrapassar os limites de seu *container* (por exemplo, especificar overflow : TextOverflow.ellipsis faz reticências serem acrescentadas ao final); textAlign permite determinar como o texto deve ser alinhado horizontalmente; textScaleFactor informa ao Flutter o número de pixels de fonte para cada unidade de pixel lógico e dessa forma dimensiona o texto.
- Para centralizarmos os 3 *strings* ao centro de Row, devemos adicionar mainAxisAlignment : MainAxisAlignment.center (propriedade) à chamada ao construtor de row.
- Caso queiramos que o segundo *string* preencha todo o espaço de row, devemos escrever: Expanded(child : Text("frase 2"))
- Expanded fará o filho preencher todo o espaço disponível. Agora, entre o primeiro e o terceiro *widgets* Text serem renderizados (usando o espaço que eles demandam, mas sem

Layout

ultrapassá-lo, já que não tentamos especificar uma largura para cada um) o que restar será preenchido pelo segundo *widget* Text.

Google *Maps*: exemplo básico

```
$ flutter create google_maps_in_flutter
Creating project google_maps_in_flutter...
[Listing of created files elided]
Wrote 127 files.

All done!
In order to run your application, type:

  $ cd google_maps_in_flutter
  $ flutter run

Your application code is in google_maps_in_flutter/lib/main.dart.
```

- Após criar o projeto Flutter, adicione o *plug-in* do Google Maps ao Flutter como uma dependência:

Google *Maps*: exemplo básico

```
$ cd google_maps_in_flutter
$ flutter pub add google_maps_flutter
Resolving dependencies...
  async 2.6.1 (2.8.2 available)
  charcode 1.2.0 (1.3.1 available)
+ flutter_plugin_android_lifecycle 2.0.3
+ google_maps_flutter 2.0.8
+ google_maps_flutter_platform_interface 2.1.1
  matcher 0.12.10 (0.12.11 available)
  meta 1.3.0 (1.7.0 available)
+ plugin_platform_interface 2.0.1
+ stream_transform 2.0.0
  test_api 0.3.0 (0.4.3 available)
Downloading google_maps_flutter 2.0.8...
Downloading flutter_plugin_android_lifecycle 2.0.3...
Changed 5 dependencies!
```

- Como a versão da *web* do *plug-in* Google Maps no Flutter ainda não é federada, adicione-o ao seu projeto:

Google *Maps*: exemplo básico

```
$ flutter pub add google_maps_flutter_web
Resolving dependencies...
  async 2.6.1 (2.8.2 available)
  charcode 1.2.0 (1.3.1 available)
+ csslib 0.17.0
+ flutter_web_plugins 0.0.0 from sdk flutter
+ google_maps 5.3.0
+ google_maps_flutter_web 0.3.0+4
+ html 0.15.0
+ js 0.6.3
+ js_wrapping 0.7.3
  matcher 0.12.10 (0.12.11 available)
  meta 1.3.0 (1.7.0 available)
+ sanitize_html 2.0.0
  test_api 0.3.0 (0.4.3 available)
Changed 8 dependencies!
```

Google *Maps*: exemplo básico

- Para usar a versão mais recente do SDK do Google Maps no iOS, é necessário ter no mínimo a versão do iOS 11 da plataforma. Modifique o ios/Podfile da seguinte maneira:

```
# Set platform to 11.0 to enable latest Google Maps SDK
platform :ios, '11.0' # Uncomment and set to 11.

# CocoaPods analytics sends network stats synchronously affecting
flutter build latency.
ENV['COCOAPODS_DISABLE_STATS'] = 'true'
```

- Para usar o SDK do Google Maps no Android, defina minSDK como 20. Modifique o android/app/build.gradle conforme exibido ao lado:

```
android {
    defaultConfig {
        // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html).
        applicationId "com.example.google_maps_in_flutter"
        minSdkVersion 20                    // Update from 16 to 20
        targetSdkVersion 30
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }
}
```

Google *Maps*: exemplo básico

- Para usar o Google Maps no seu *app* do Flutter, é necessário configurar um projeto de API com a Plataforma Google Maps, seguindo as instruções de como usar chaves de API no SDK do Maps para Android (https://developers.google.com/maps/documentation/android-sdk/signup?hl=pt-br), no SDK do Maps para iOS (https://developers.google.com/maps/documentation/ios-sdk/get-api-key?hl=pt-br) e no SDK da API Maps JavaScript (https://developers.google.com/maps/documentation/javascript/get-api-key?hl=pt-br).
- Com as chaves de API em mãos, execute as etapas a seguir para configurar os aplicativos Android e iOS:

i. Para adicionar uma chave de API ao *app* Android, edite o arquivo AndroidManifest.xml em android/app/src/main. Adicione uma única entrada meta-data contendo a chave de API criada na etapa anterior dentro da *tag* application.

Google *Maps*: exemplo básico

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.google_maps_in_flutter">
    <application
        android:label="google_maps_in_flutter"
        android:icon="@mipmap/ic_launcher">

        <!-- TODO: Add your Google Maps API key here -->
        <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="YOUR-KEY-HERE"/>

        <activity
            android:name=".MainActivity"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"
            android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDir
ection|fontScale|screenLayout|density|uiMode"
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <meta-data
              android:name="io.flutter.embedding.android.NormalTheme"
              android:resource="@style/NormalTheme"
              />
```

Google *Maps*: exemplo básico

```
            <meta-data
              android:name="io.flutter.embedding.android.SplashScreenDrawable"
              android:resource="@drawable/launch_background"
              />
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <meta-data
            android:name="flutterEmbedding"
            android:value="2" />
    </application>
</manifest>
```

Google *Maps*: exemplo básico

ii. Para adicionar uma chave de API ao *app* iOS, edite o arquivo AppDelegate.swift em ios/Runner. Ao contrário do Android, adicionar uma chave de API no iOS exige mudanças no código-fonte do *app* Runner. O AppDelegate é o *Singleton* principal que faz parte do processo de inicialização do *app*.

iii. Faça duas alterações nesse arquivo. Primeiro, adicione uma instrução #import para extrair os cabeçalhos do Google Maps e, em seguida, chame o método provideAPIKey() do *Singleton* GMSServices. Essa chave de API permite que o Google Maps exiba corretamente os blocos de mapas.

Google *Maps*: exemplo básico

```swift
//ios/Runner/AppDelegate.swift
import UIKit
import Flutter
import GoogleMaps  // Add this import

@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
  ) -> Bool {
    GeneratedPluginRegistrant.register(with: self)

    // TODO: Add your Google Maps API key
    GMSServices.provideAPIKey("YOUR-API-KEY")

    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

Google *Maps*: exemplo básico

iv. Para adicionar uma chave de API ao *app* da *web*, edite o arquivo index.html em web. Adicione uma referência ao *script* de JavaScript do Google Maps na seção <head> incluindo sua chave de API.

```
<head>
  <base href="/">

  <meta charset="UTF-8">
  <meta content="IE=Edge" http-equiv="X-UA-Compatible">
  <meta name="description" content="A new Flutter project.">

  <!-- iOS meta tags & icons -->
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="apple-mobile-web-app-title" content="google_maps_in_flutter">
  <link rel="apple-touch-icon" href="icons/Icon-192.png">

  <!-- TODO: Add your Google Maps API key here -->
  <script src="https://maps.googleapis.com/maps/api/js?key=YOUR-KEY-HERE"></script>
  <title>google_maps_in_flutter</title>
  <link rel="manifest" href="manifest.json">

</head>
```

Google *Maps*: exemplo básico

v. Agora, é hora de mostrar um mapa na tela. Atualize lib/main.dart desta maneira:

```dart
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';

void main() => runApp(const MyApp());

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  late GoogleMapController mapController;

  final LatLng _center = const LatLng(45.521563, -122.677433);

  void _onMapCreated(GoogleMapController controller) {
    mapController = controller;
  }
```

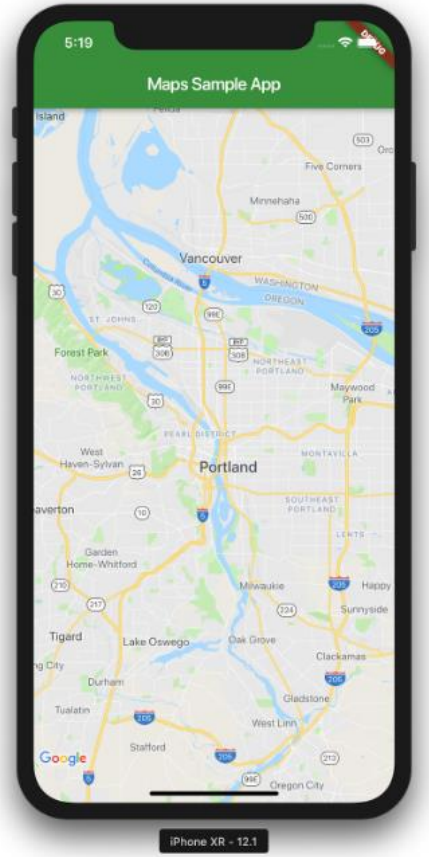
Google *Maps*: exemplo básico

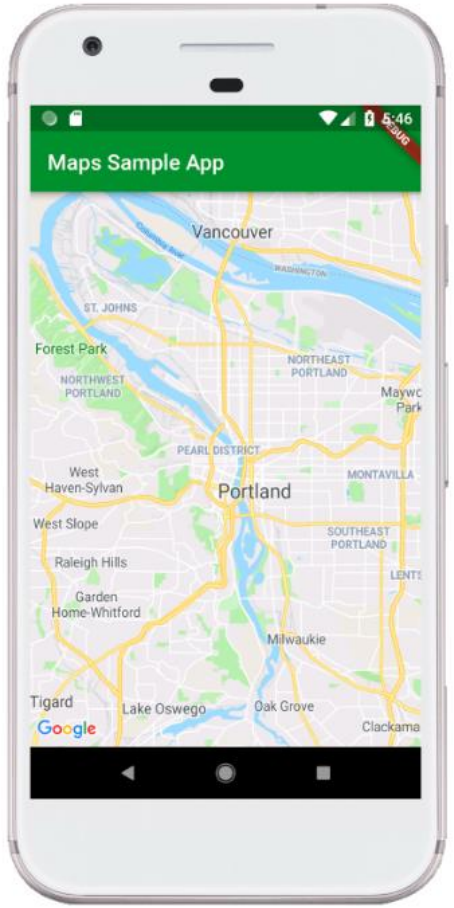
```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Maps Sample App'),
          backgroundColor: Colors.green[700],
        ),
        body: GoogleMap(
          onMapCreated: _onMapCreated,
          initialCameraPosition: CameraPosition(
            target: _center,
            zoom: 11.0,
          ),
        ),
      ),
    );
  }
}
```

Google *Maps*: exemplo básico

vi. Execute o app do Flutter no iOS ou Android para ter uma única visualização de mapa, centralizada em Portland. Outra opção é executar um emulador do Android ou um simulador de iOS. Você pode modificar o centro do mapa para representar sua cidade ou algum lugar importante para você.

```
$ flutter run
```





Google *Maps*: exemplo básico

- O Google tem muitos escritórios no mundo todo, desde a América do Norte, América Latina, Europa e Ásia-Pacífico até a África e o Oriente Médio. Se você investigar esses mapas, vai ver que eles têm um *endpoint* de API fácil de usar para fornecer as informações do local dos escritórios no formato JSON. Nesta etapa, você irá colocar esses escritórios no mapa, além de usar a geração de código para analisar o JSON.
- Adicione três novas dependências do Flutter ao projeto da seguinte forma: Primeiro, adicione o pacote http para facilitar as solicitações HTTP.

```
$ flutter pub add http
Resolving dependencies...
  async 2.8.1 (2.8.2 available)
+ http 0.13.3
+ http_parser 4.0.0
  matcher 0.12.10 (0.12.11 available)
+ pedantic 1.11.1
  test_api 0.4.2 (0.4.3 available)
Changed 3 dependencies!
```

Google *Maps*: exemplo básico

- Em seguida, adicione json_serializing para declarar a estrutura de objetos e representar documentos JSON.

```
$ flutter pub add json_serializable
Resolving dependencies...
+ _fe_analyzer_shared 25.0.0
+ analyzer 2.2.0
+ args 2.2.0
  async 2.8.1 (2.8.2 available)
+ build 2.1.0
+ build_config 1.0.0
+ checked_yaml 2.0.1
+ cli_util 0.3.3
+ convert 3.0.1
+ crypto 3.0.1
+ dart_style 2.0.3
+ file 6.1.2
+ glob 2.0.1
+ json_annotation 4.1.0
+ json_serializable 5.0.0
+ logging 1.0.1
  matcher 0.12.10 (0.12.11 available)
```

```
+ package_config 2.0.0
+ pub_semver 2.0.0
+ pubspec_parse 1.0.0
+ source_gen 1.1.0
+ source_helper 1.2.1
  test_api 0.4.2 (0.4.3 available)
+ watcher 1.0.0
+ yaml 3.1.0
Downloading analyzer 2.2.0...
Downloading _fe_analyzer_shared 25.0.0...
Changed 22 dependencies!
```

Google *Maps*: exemplo básico

- Por fim, adicione build_runner como uma dependência do tempo de desenvolvimento.

```
$ flutter pub add --dev build_runner
Resolving dependencies...
  async 2.8.1 (2.8.2 available)
+ build_daemon 3.0.0
+ build_resolvers 2.0.4
+ build_runner 2.1.1
+ build_runner_core 7.1.0
+ built_collection 5.1.0
+ built_value 8.1.2
+ code_builder 4.1.0
+ fixnum 1.0.0
+ frontend_server_client 2.1.2
+ graphs 2.0.0
+ http_multi_server 3.0.1
+ io 1.0.3
+ js 0.6.3
  matcher 0.12.10 (0.12.11 available)
+ mime 1.0.0
+ pool 1.5.0
```

```
+ shelf 1.2.0
+ shelf_web_socket 1.0.1
  test_api 0.4.2 (0.4.3 available)
+ timing 1.0.0
+ web_socket_channel 2.1.0
Changed 19 dependencies!
```

Google *Maps*: exemplo básico

- Observe que os dados JSON retornados do *endpoint* de API têm uma estrutura regular. Seria interessante gerar o código para organizar esses dados em objetos que você pode usar no código.
- No diretório lib/src, crie um arquivo locations.dart e descreva a estrutura dos dados JSON retornados da seguinte forma:

```dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:json_annotation/json_annotation.dart';
import 'package:flutter/services.dart' show rootBundle;

part 'locations.g.dart';

@JsonSerializable()
class LatLng {
  LatLng({
    required this.lat,
    required this.lng,
  });

  factory LatLng.fromJson(Map<String, dynamic> json) => _$LatLngFromJson(json);
  Map<String, dynamic> toJson() => _$LatLngToJson(this);
```

Google *Maps*: exemplo básico

```
  final double lat;
  final double lng;
}

@JsonSerializable()
class Region {
  Region({
    required this.coords,
    required this.id,
    required this.name,
    required this.zoom,
  });

  factory Region.fromJson(Map<String, dynamic> json) => _$RegionFromJson(json);
  Map<String, dynamic> toJson() => _$RegionToJson(this);

  final LatLng coords;
  final String id;
  final String name;
  final double zoom;
}

@JsonSerializable()
```

Google *Maps*: exemplo básico

```
class Office {
  Office({
    required this.address,
    required this.id,
    required this.image,
    required this.lat,
    required this.lng,
    required this.name,
    required this.phone,
    required this.region,
  });

  factory Office.fromJson(Map<String, dynamic> json) => _$OfficeFromJson(json);
  Map<String, dynamic> toJson() => _$OfficeToJson(this);

  final String address;
  final String id;
  final String image;
  final double lat;
  final double lng;
  final String name;
  final String phone;
  final String region;
}
```

Google *Maps*: exemplo básico

```
@JsonSerializable()
class Locations {
  Locations({
    required this.offices,
    required this.regions,
  });

  factory Locations.fromJson(Map<String, dynamic> json) =>
      _$LocationsFromJson(json);
  Map<String, dynamic> toJson() => _$LocationsToJson(this);

  final List<Office> offices;
  final List<Region> regions;
}

Future<Locations> getGoogleOffices() async {
  const googleLocationsURL = 'https://about.google/static/data/locations.json';

  // Retrieve the locations of Google offices
  try {
    final response = await http.get(Uri.parse(googleLocationsURL));
    if (response.statusCode == 200) {
      return Locations.fromJson(json.decode(response.body));
```

Google *Maps*: exemplo básico

```
    }
  } catch (e) {
    print(e);
  }

  // Fallback for when the above HTTP request fails.
  return Locations.fromJson(
    json.decode(
      await rootBundle.loadString('assets/locations.json'),
    ),
  );
}
```

Google *Maps*: exemplo básico

- Depois de adicionar esse código, o ambiente de desenvolvimento integrado exibirá alguns destaques em vermelho, já que ele faz referência a um arquivo irmão que não existe, o locations.g.dart.. Esse arquivo gerado é convertido entre estruturas JSON não digitadas e objetos nomeados. Crie-o executando o build_runner:

```
$ flutter pub run build_runner build --delete-conflicting-outputs
[INFO] Generating build script...
[INFO] Generating build script completed, took 357ms

[INFO] Creating build script snapshot......
[INFO] Creating build script snapshot... completed, took 10.5s

[INFO] There was output on stdout while compiling the build script snapshot, run with `--verbose` to see it (you will
need to run a `clean` first to re-snapshot).

[INFO] Initializing inputs
[INFO] Building new asset graph...
[INFO] Building new asset graph completed, took 646ms

[INFO] Checking for unexpected pre-existing outputs....
[INFO] Deleting 1 declared outputs which already existed on disk.
[INFO] Checking for unexpected pre-existing outputs. completed, took 3ms
```

Google *Maps*: exemplo básico

```
[INFO] Running build...
[INFO] Generating SDK summary...
[INFO] 3.4s elapsed, 0/3 actions completed.
[INFO] Generating SDK summary completed, took 3.4s

[INFO] 4.7s elapsed, 2/3 actions completed.
[INFO] Running build completed, took 4.7s

[INFO] Caching finalized dependency graph...
[INFO] Caching finalized dependency graph completed, took 36ms

[INFO] Succeeded after 4.8s with 2 outputs (7 actions)
```

Google *Maps*: exemplo básico

- Agora, seu código será analisado novamente sem erros. Em seguida, precisamos adicionar o arquivo substituto locations.json usado na função getGoogleOffices. Uma das razões para incluir esse substituto é que os dados estáticos carregados nesta função são veiculados sem cabeçalhos CORS* e, portanto, não são carregados em um navegador da *web*. Os *apps* do Flutter para Android e iOS não precisam de cabeçalhos CORS, mas o acesso a dados móveis pode ser complicado.
- Acesse https://about.google/static/data/locations.json no navegador e salve o conteúdo no diretório de assets. Outra opção é usar a linha de comando da seguinte forma:

```
$ mkdir assets
$ cd assets
$ curl -o locations.json https://about.google/static/data/locations.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 30348  100 30348    0     0  75492      0 --:--:-- --:--:-- --:--:-- 75492
```

- Agora que você já fez o *download* do arquivo de assets, adicione-o à seção do Flutter do arquivo pubspec.yaml.

* CORS (*Cross-origin Resource Sharing*) é um mecanismo usado para adicionar cabeçalhos HTTP que informam aos navegadores para permitir que uma aplicação *web* seja executada em uma origem e acesse recursos de outra origem diferente. Esse tipo de ação é chamada de requisição *cross-origin* HTTP.

Google *Maps*: exemplo básico

```
//pubspec.yaml
flutter:
  uses-material-design: true

  assets:
    - assets/locations.json
```

- Modifique o arquivo main.dart para solicitar os dados do mapa e use as informações retornadas para adicionar os escritórios ao mapa:

```
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'src/locations.dart' as locations;

void main() {
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  _MyAppState createState() => _MyAppState();
}
```

Google *Maps*: exemplo básico

```
class _MyAppState extends State<MyApp> {
  final Map<String, Marker> _markers = {};
  Future<void> _onMapCreated(GoogleMapController controller) async {
    final googleOffices = await locations.getGoogleOffices();
    setState(() {
      _markers.clear();
      for (final office in googleOffices.offices) {
        final marker = Marker(
          markerId: MarkerId(office.name),
          position: LatLng(office.lat, office.lng),
          infoWindow: InfoWindow(
            title: office.name,
            snippet: office.address,
          ),
        );
        _markers[office.name] = marker;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

Google *Maps*: exemplo básico

```
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Google Office Locations'),
          backgroundColor: Colors.green[700],
        ),
        body: GoogleMap(
          onMapCreated: _onMapCreated,
          initialCameraPosition: const CameraPosition(
            target: LatLng(0, 0),
            zoom: 2,
          ),
          markers: _markers.values.toSet(),
        ),
      ),
    );
  }
}
```

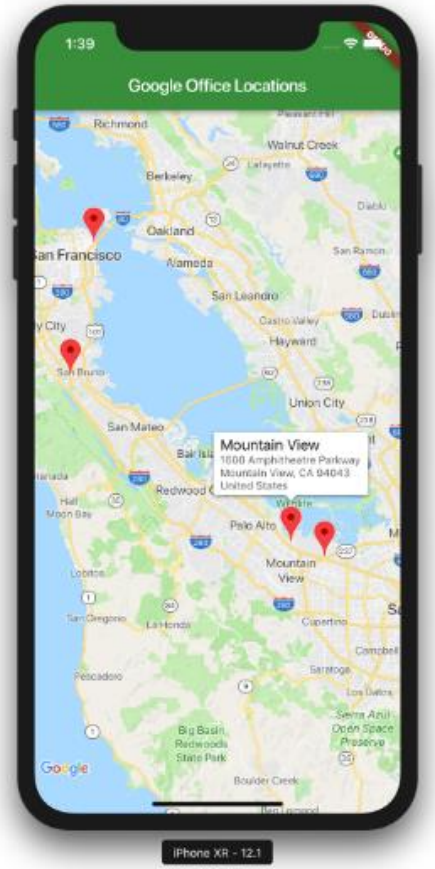
Google *Maps*: exemplo básico

- Esse código executa várias operações:

i. Em _onMapCreated, ele usa o código de análise JSON da etapa anterior, aguardando (await) até que ele seja carregado. Em seguida, usará os dados retornados para criar marcadores (Marker) dentro de um *callback* setState(). Quando o *app* recebe novos marcadores, o setState sinaliza o Flutter para repintar a tela, exibindo os locais dos escritórios.
ii. Os marcadores são armazenados em um Map associado ao *widget* GoogleMap. Isso vincula os marcadores ao mapa correto. Obviamente, você pode ter vários mapas e exibir marcadores diferentes em cada um.

Google *Maps*: exemplo básico

- Há muitas coisas interessantes que podem ser adicionadas a partir de agora. Por exemplo, você pode adicionar uma visualização dos escritórios em lista, que se move e amplia o mapa quando o usuário clica em um deles



Dúvidas!?