



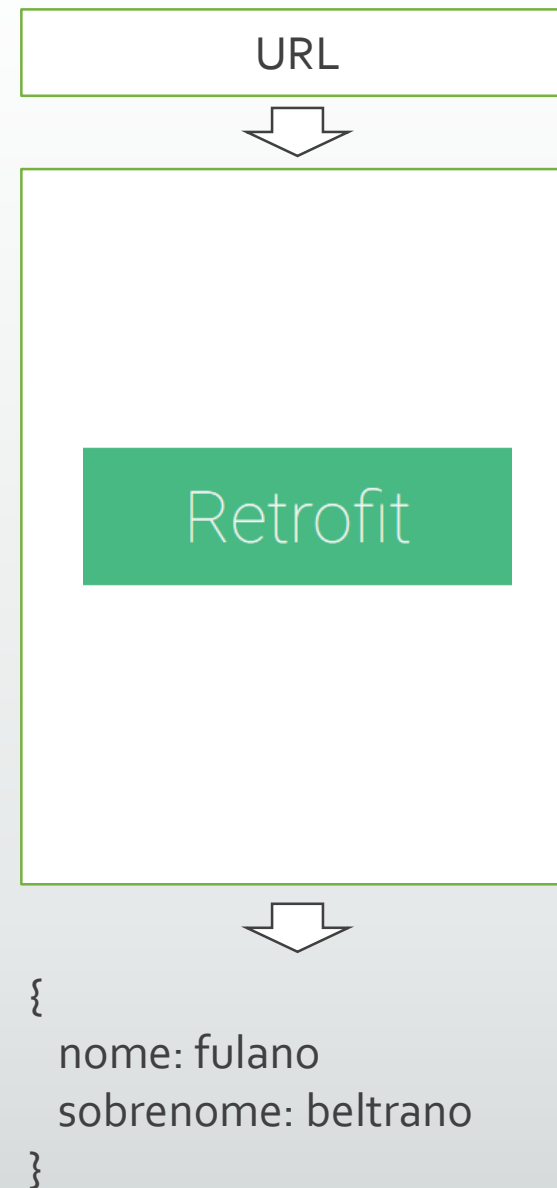
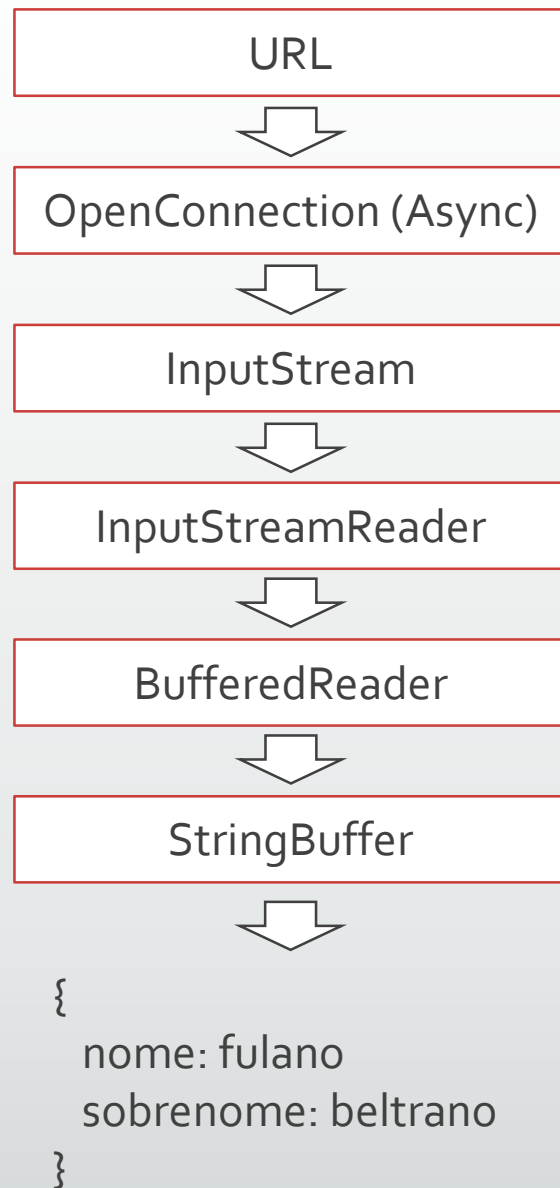
# Biblioteca Retrofit

Dispositivos Móveis

Prof. Edson Mota, PhD, MSc, PMP

Quais os **passos** para  
**consumir** uma API?





# O que é Retrofit? - <https://square.github.io/retrofit/>

- Retrofit é uma biblioteca para Android que facilita a comunicação com APIs RESTful por meio da conversão de chamadas HTTP em objetos Java.
- Esse processo ocorre por meio da definição de uma interface Java que representa a API RESTful a ser consumida e o uso dessa mesma interface para fazer chamadas HTTP para uma certa API.
- Retrofit simplifica o processo de comunicação HTTP, na medida em que torna o acesso às APIs remotas em um processo declarativo, ocultando a complexidade das chamadas.



# Alguns Benefícios

- **Abstração da camada de rede**

- Permite que os desenvolvedores escrevam código mais limpo, pois abstrai muitos detalhes da comunicação e construção de requisições HTTP.

- **Melhor legibilidade do código**

- Como as chamadas de API são definidas em interfaces Java, o código que faz uso do Retrofit simplifica e torna o processo de desenvolvimento mais eficiente.

- **Flexibilidade**

- Retrofit oferece suporte para várias formas de autenticação, serialização de dados, conversão de dados da resposta e manipulação de erros de rede.

- **Melhoria no desempenho**

- Retrofit é projetado para ser eficiente em termos de recursos, com pouca sobrecarga, o que resulta em um melhor desempenho do aplicativo.

- Entre outros

# OkHttp



- A biblioteca Retrofit se utiliza de uma outra biblioteca de comunicação denominada **OkHttp**
- Trata-se de uma biblioteca cliente HTTP de código aberto que é usada pelo Retrofit 2.x como sua camada de transporte HTTP padrão.
- OkHttp é responsável por fazer as chamadas HTTP de fato, enviando solicitações e recebendo respostas para APIs RESTful.
- Além disso, o OkHttp fornece recursos avançados, como **armazenamento em cache, reutilização de conexão**, redirecionamento automático, suporte a protocolo HTTP/2 e muitos **outros recursos** que tornam o processo de comunicação com APIs RESTful mais eficiente e confiável.

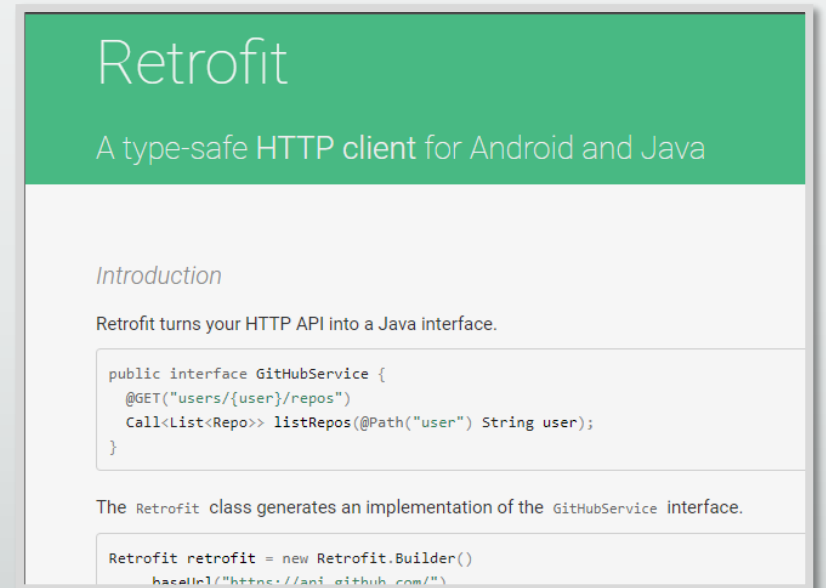
# Por trás dos Panos



# O site oficial oferece uma excelente documentação da biblioteca

Tem **dúvida?** Use a **doc!!**

<https://square.github.io/retrofit/> →





# Vamos Programar?

ViaCEP - Webservice CEP e IBGE x +

viacep.com.br

# VIA CEP

Consulte CEPs de todo o Brasil

Procurando um [webservice](#) gratuito e de alto desempenho para consultar Códigos de Endereçamento Postal. Utilize nosso serviço, melhore a qualidade de suas aplicações web e colabore para manter esta base de dados.

## Acessando o webservice de CEP

Para acessar o webservice, um CEP no formato `XXXX-XXXX` é necessário. Após o CEP, deve ser fornecido o tipo de retorno desejado.

Exemplo de pesquisa por CEP:

[viacep.com.br/ws/01001000/json/](https://viacep.com.br/ws/01001000/json/)

```
{
  "cep": "41650-010",
  "logradouro": "Avenida Orlando Gomes",
  "complemento": "",
  "bairro": "Piatã",
  "localidade": "Salvador",
  "uf": "BA",
  "ibge": "2927408",
  "gia": "",
  "ddd": "71",
  "siafi": "3849"
}
```



Retrofit

square.github.io/retrofit/

# Retrofit

Download

The source code to the Retrofit, its samples, and this website is [available on GitHub](#).

## MAVEN

```
<dependency>
  <groupId>com.squareup.retrofit2</groupId>
  <artifactId>retrofit</artifactId>
  <version>(insert latest version)</version>
</dependency>
```

## GRADLE

```
implementation 'com.squareup.retrofit2:retrofit:(insert latest version)'
```

Retrofit requires at minimum Java 8+ or Android API 21+.


## R8 / PROGUARD

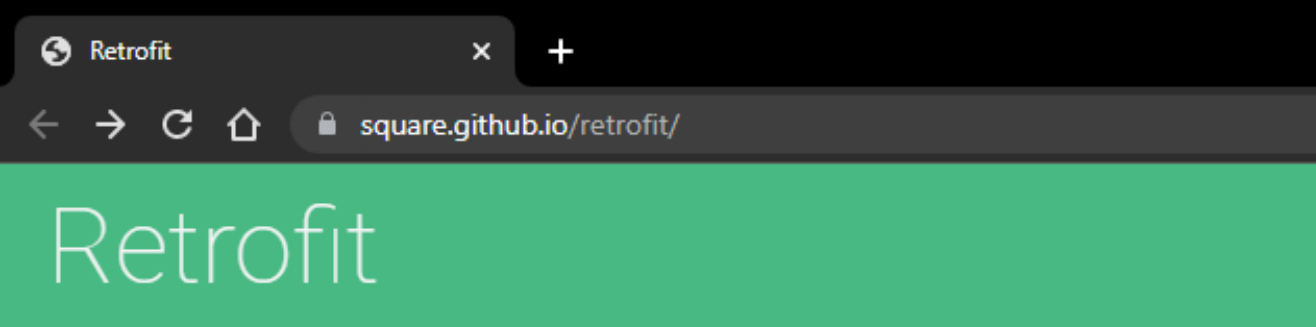
If you are using R8 the shrinking and obfuscation rules are included automatically.

ProGuard users must manually add the options from [retrofit2.pro](#).

You might also need rules for [OkHttp](#) and [Okio](#) which are dependencies of this library.

Configuração da biblioteca  
Retrofit que será incorporada  
ao **Gradle Module**





## CONVERTERS

By default, Retrofit can only deserialize HTTP bodies into OkHttp's `ResponseBody` type and it can accept its `RequestBody` type for `@Body`.

Converters can be added to support other types. Six sibling modules adapt popular serialization libraries for your convenience.

- **Gson:** `com.squareup.retrofit2:converter-gson`
- **Jackson:** `com.squareup.retrofit2:converter-jackson`
- **Moshi:** `com.squareup.retrofit2:converter-moshi`
- **Protobuf:** `com.squareup.retrofit2:converter-protobuf`
- **Wire:** `com.squareup.retrofit2:converter-wire`
- **Simple XML:** `com.squareup.retrofit2:converter-simplexml`
- **JAXB:** `com.squareup.retrofit2:converter-jaxb`
- **Scalars (primitives, boxed, and String):** `com.squareup.retrofit2:converter-scalars`

Here's an example of using the `GsonConverterFactory` class to generate an implementation of `GitHubService` interface which uses Gson for its deserialization.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

GitHubService service = retrofit.create(GitHubService.class);
```

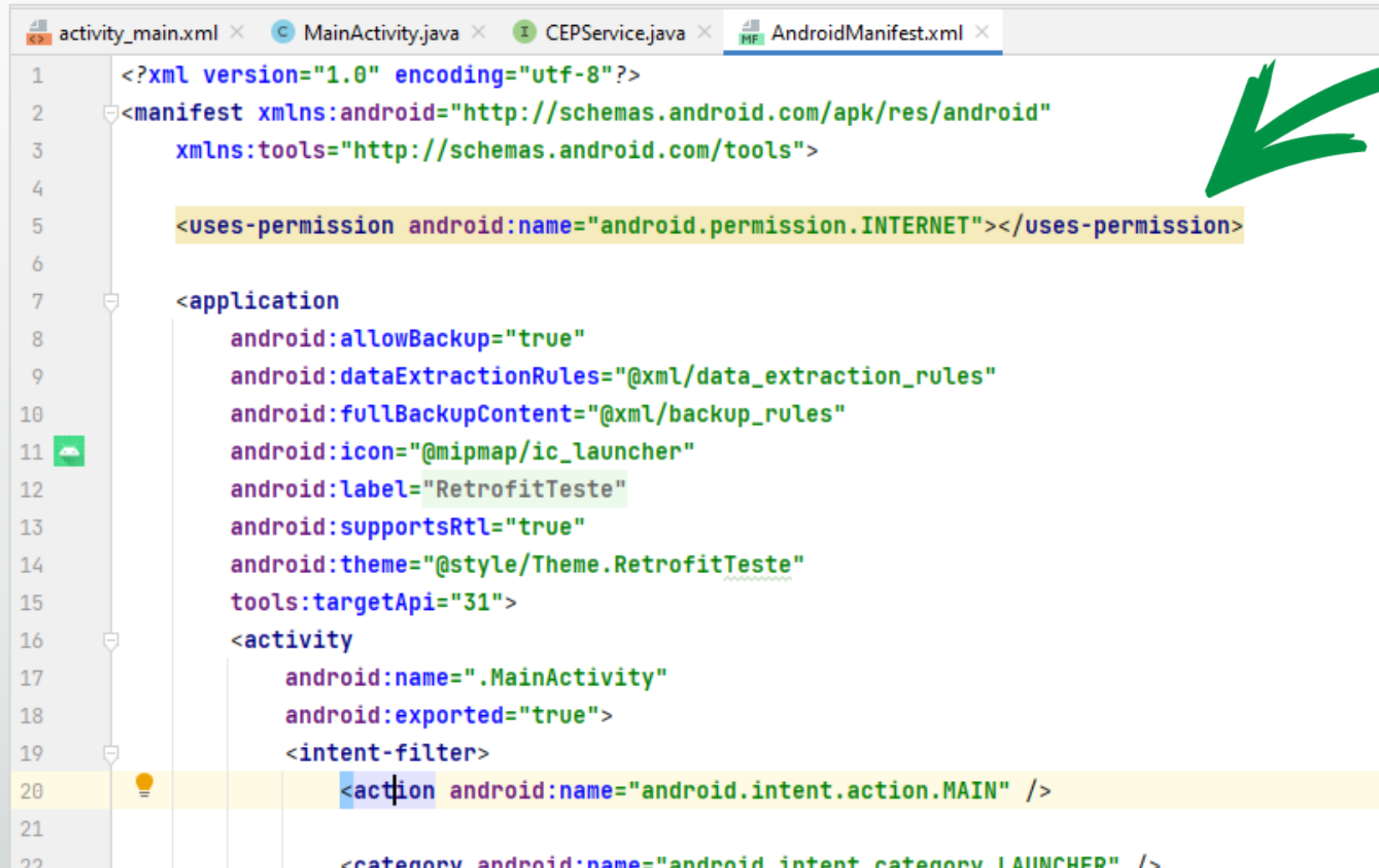


Conversor JSON utilizado pela Retrofit para serializar dados em objetos/model

# Gradle Module

```
dependencies {  
  
    implementation 'androidx.appcompat:appcompat:1.6.1'  
    implementation 'com.google.android.material:material:1.8.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'  
    testImplementation 'junit:junit:4.13.2'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'  
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
}
```

# Android Manifest



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         xmlns:tools="http://schemas.android.com/tools">
4
5         <uses-permission android:name="android.permission.INTERNET"></uses-permission>
6
7         <application
8             android:allowBackup="true"
9             android:dataExtractionRules="@xml/data_extraction_rules"
10            android:fullBackupContent="@xml/backup_rules"
11            android:icon="@mipmap/ic_launcher"
12            android:label="RetrofitTeste"
13            android:supportsRtl="true"
14            android:theme="@style/Theme.RetrofitTeste"
15            tools:targetApi="31">
16             <activity
17                 android:name=".MainActivity"
18                 android:exported="true">
19                 <intent-filter>
20                     <action android:name="android.intent.action.MAIN" />
21
22                     <category android:name="android.intent.category.LAUNCHER" />
```

```
10 public class MainActivity extends AppCompatActivity {
11
12     1 usage
13     private TextView txtRetornaDados;
14     2 usages
15     private Button btGetData;
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_main);
21
22         txtRetornaDados = findViewById(R.id.txtRetornaDados);
23         btGetData = findViewById(R.id.btGetData);
24
25         btGetData.setOnClickListener(new View.OnClickListener() {
26             @Override
27             public void onClick(View v) {
28                 RecuperaCEPRetrify();
29             }
30         });
31
32     }
33     1 usage
34     private void RecuperaCEPRetrify(){
35
36     }
37 }
```

# Retrofit

- Precisamos declarar um objeto Retrofit

```
retrofit = new Retrofit.Builder()  
    .baseUrl("https://viacep.com.br/ws/")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();
```

- BaseUrl: Deve indicar a URL da API
- GsonConverterFactory: Habilita o conversor p/ serialização dos dados em objetos.



# URL x API

## Url completa da API

`https://viacep.com.br/ws/41650010/json/`

URL Base

Parâmetros



- `baseUrl ("https://viacep.com.br/ws/")`

```

13  public class MainActivity extends AppCompatActivity {
14
15      1 usage
      private TextView txtRetornaDados;
16      2 usages
      private Button btGetData;
17      1 usage
      private Retrofit retrofit;
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.activity_main);
23
24          txtRetornaDados = findViewById(R.id.txtRetornaDados);
25          btGetData = findViewById(R.id.btGetData);
26
27          //Configura o objeto retrofit.
28          retrofit = new Retrofit.Builder()
29              .baseUrl("https://viacep.com.br/ws/")
30              .addConverterFactory(GsonConverterFactory.create())
31              .build();
32
33          btGetData.setOnClickListener(new View.OnClickListener() {

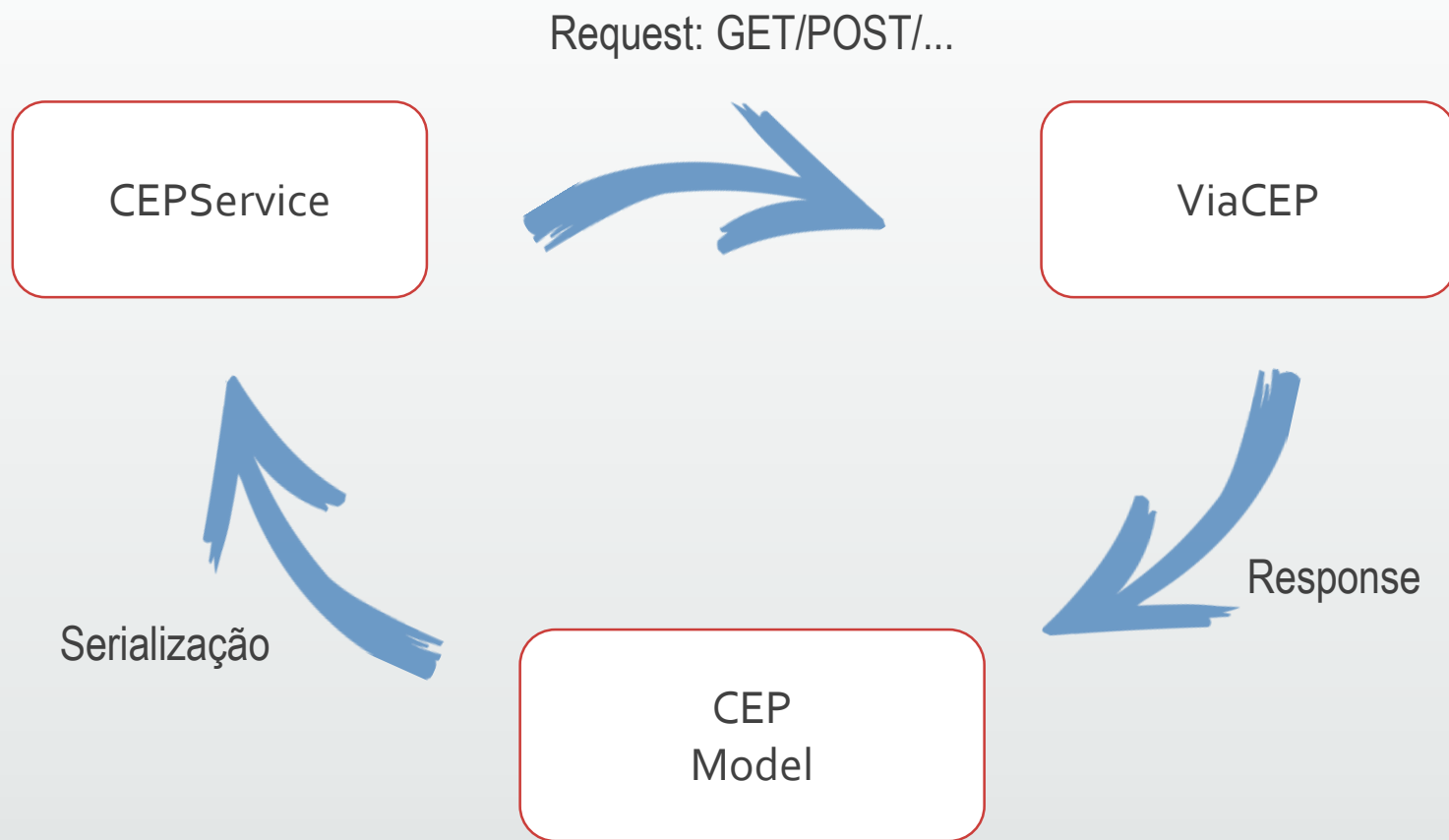
```

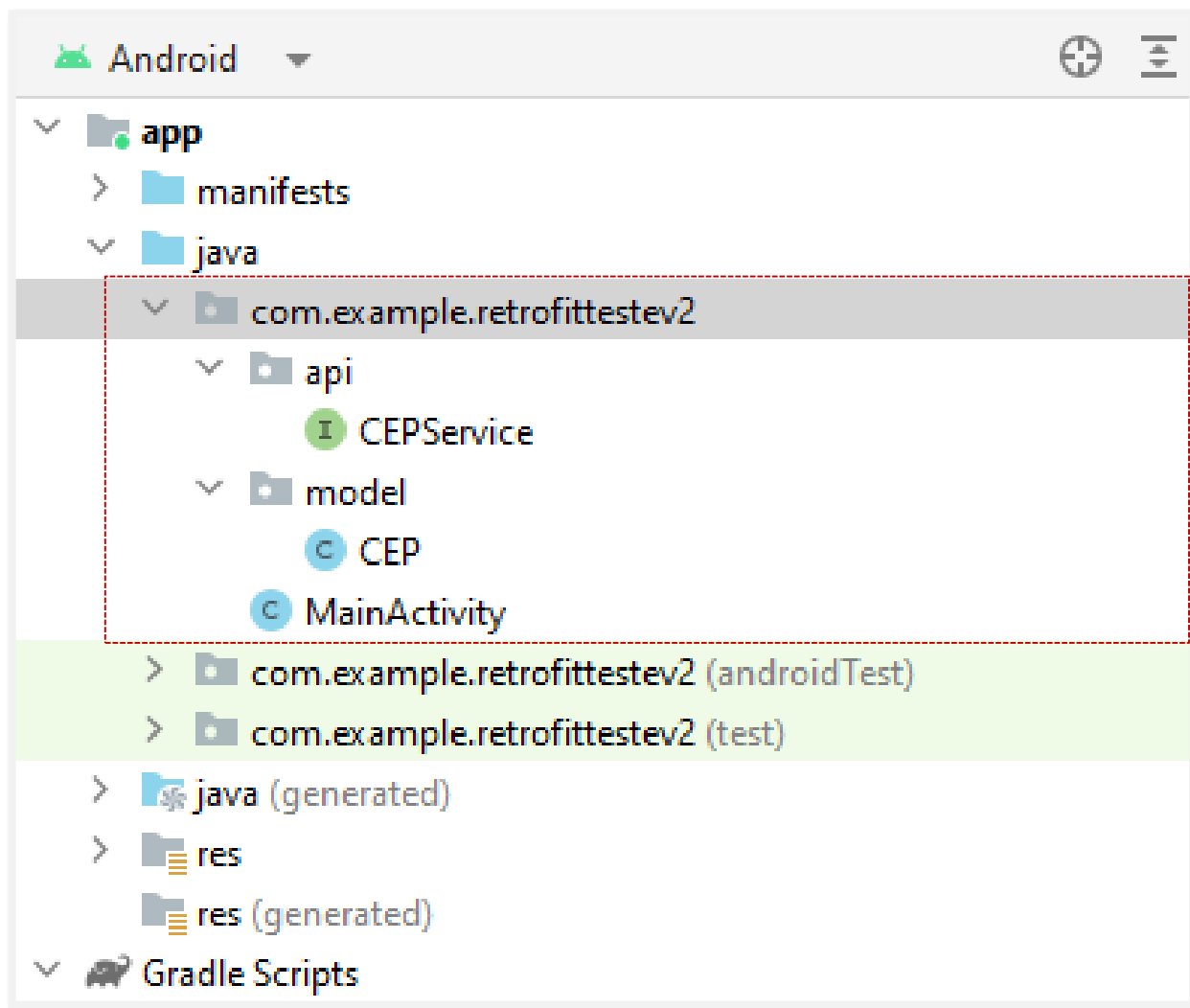
Declarando o objeto  
Retrofit



# Estrutura de Dados e Mecanismos de Acesso

- Quando usamos a biblioteca Retrofit precisamos de dois elementos principais:
  - Uma Interface **Service**
    - Responsável por direcionar as requisições para a rota correta.
  - Uma Classe **Model**
    - Responsável por receber os dados e serializá-los em objetos.





Estrutura dos  
packages



```
1 package com.example.retrofittestev2.api;
2
3 import com.example.retrofittestev2.model.CEP;
4
5 import retrofit2.Call;
6 import retrofit2.http.GET;
7
8 public interface CEPService {
9
10     //Rota da API
11     @GET("41650010/json/")
12     Call<CEP> RecuperaCep();
13
14 }
```

Implementação do  
CEPService



```

1 package com.example.retrofittestev2.model;
2
3 public class CEP {
4     2 usages
4     private String cep;
5     2 usages
5     private String logradouro;
6     2 usages
6     private String complemento;
7     2 usages
7     private String bairro;
8     2 usages
8     private String localidade;
9     2 usages
9     private String uf;
10
11     public String getCep() {
12         return cep;
13     }

```

Implementação do  
Model: CEP



Getters e Setters foram omitidos da imagem



Agora que temos a  
estrutura, podemos  
programar a  
recuperação dos dados



1 usage

```
44 private void RecuperaCEPRetrifyfit(){
```

```
46     CEPService cepservice = retrofit.create(CEPService.class);
```

```
47     Call<CEP> call = cepservice.RecuperaCep();
```

```
49     call.enqueue(new []);
```

```
52 }
```

```
53 }
```

I Callback<CEP>{...} (retrofit2)

C MainActivity com.example.retrofittestev2

C CEP com.example.retrofittestev2.model

I CEPService com.example.retrofittestev2.api

C R com.example.retrofittestev2

C Manifest com.example.retrofittestev2

C BuildConfig com.example.retrofittestev2

C AppCompatActivity androidx.appcompat.app

C Bundle android.os

C Button android.widget

C TextView android.widget

C View android.view

Press Enter to insert, Guia to replace [Next Tip](#)

```
46 private void RecuperaCEPReetrofit(){
```

```
47  
48     CEPService cepservice = retrofit.create(CEPService.class);
```

```
49     Call<CEP> call = cepservice.RecuperaCep();
```

```
50  
51     call.enqueue(new Callback<CEP>() {
```

```
52         @Override
```

```
53         public void onResponse(Call<CEP> call, Response<CEP> response) {
```

```
54         |
```

```
55     }
```

```
56  
57     @Override
```

```
58     public void onFailure(Call<CEP> call, Throwable t) {
```

```
59  
60     }
```

```
61     });
```



Chamada  
assíncrona

```

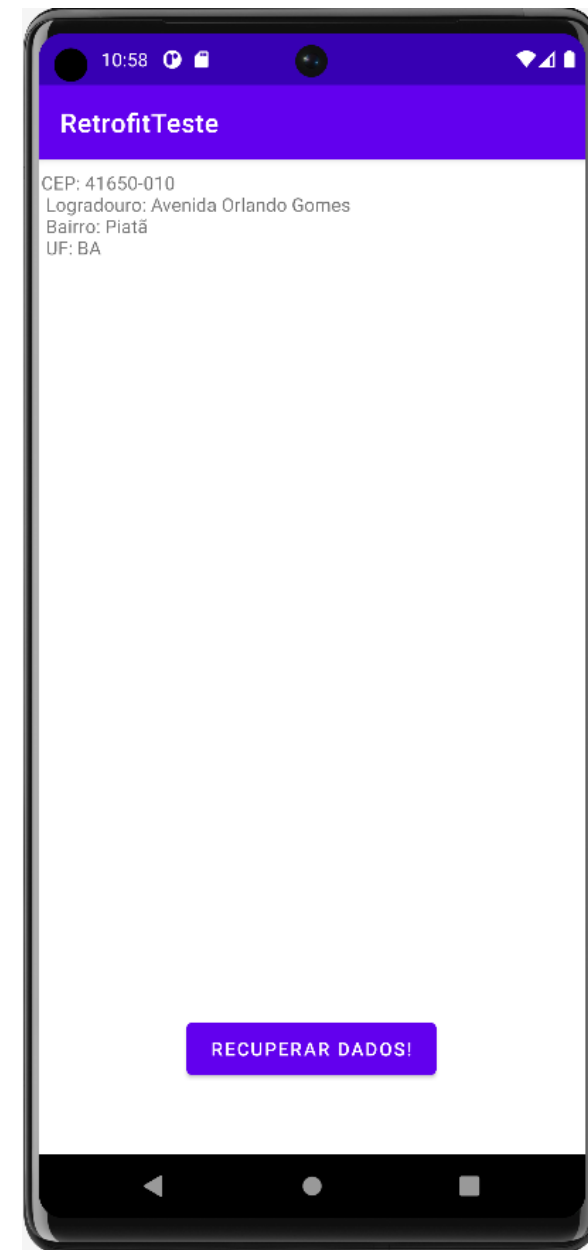
46 private void RecuperaCEPReetrofit(){
47
48     CEPService cepservice = retrofit.create(CEPService.class);
49     Call<CEP> call = cepservice.RecuperaCep();
50
51     call.enqueue(new Callback<CEP>() {
52         @Override
53         public void onResponse(Call<CEP> call, Response<CEP> response) {
54
55             if (response.isSuccessful()) {
56                 CEP cep = response.body();
57
58                 txtRetornaDados.setText(
59                     "CEP: " + cep.getCep()
60                     + "\n Logradouro: " + cep.getLogradouro()
61                     + "\n Bairro: " + cep.getBairro()
62                     + "\n UF: " + cep.getUf()
63                 );
64             }
65         }
66         @Override
67         public void onFailure(Call<CEP> call, Throwable t) {
68
69         }
70     });
71 }

```

Recuperando os  
dados e mostrando  
o TextView



```
{  
  "cep": "41650-010",  
  "logradouro": "Avenida Orlando Gomes",  
  "complemento": "",  
  "bairro": "Piatã",  
  "localidade": "Salvador",  
  "uf": "BA",  
  "ibge": "2927408",  
  "gia": "",  
  "ddd": "71",  
  "siafi": "3849"  
}
```



# Parametrizando a Interface Service

# Refatorando a Interface

- Utilizando APIs, é comum termos a necessidade de enviarmos certos parâmetros que indicarão à API o tipo de dados que desejamos obter.
- Essa parametrização pode ser feita realizando uma pequena refatoração na interface Service.

```
public interface CEPService {  
  
    //Rota da API  
    1 usage  
    @GET("{cep}/json/")  
    Call<CEP> RecuperaCep(@Path("cep") String cep);  
  
}
```

Agora, precisamos passar o parâmetro para a interface.

```
private void RecuperaCEPReetrofit(){
```

```
    CEPService cepservice = retrofit.create(CEPService.class);
```

```
    Call<CEP> call = cepservice.RecuperaCep("41650010");
```

```
    call.enqueue(new Callback<CEP>() {
```

```
        @Override
```

```
        public void onResponse(Call<CEP> call, Response<CEP> response) {
```

```
            if (response.isSuccessful()) {
```



# Revisão

- Aprimore a experiência do APP criando uma interface mais elaborada na qual seja possível informar um CEP e receber de volta os dados da API (Via-CEP).
- Inclua um outro meio de pesquisa no qual seja possível informar o endereço ou partes de um endereço e receber de volta os dados da API.
  - **Identifiquem na documentação da API como essa consulta funciona.**
- Dica....
  - Para criar um serviço que retorne uma lista, pode-se utilizar

```
@GET("{parâmetros}/json/")  
Call<List<CEP>> GetByEndereco(@Path("parâmetros") String parâmetros);
```

# JsonObject

5 usages

```
public interface CEPService {
```

```
    //Rota da API
```

1 usage

```
@GET("{cep}/json/")
```

```
Call<CEP> RecuperaCep(@Path("cep") String cep);
```

```
@GET("{cep}/json/")
```

```
Call<JsonObject> RecuperaCepJsonObject(@Path("cep") String cep);
```

1 usage

```
private void RecuperaCEPReetrofitJsonObject() {  
    CEPService cepservice = retrofit.create(CEPService.class);  
    Call<JsonObject> call = cepservice.RecuperaCepJsonObject("41650010");  
  
    call.enqueue(new Callback<JsonObject>() {  
        @Override  
        public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {  
            if (response.isSuccessful()) {  
                JsonObject jsonObject = response.body();  
                String log = jsonObject.get("logradouro").getString();  
                txtRetornaDados.setText(log);  
            }  
        }  
    });  
  
    @Override  
    public void onFailure(Call<JsonObject> call, Throwable t) {  
    }  
});
```

Recebe um  
objeto JSON  
Válido

Obtém a chave, no entanto,  
se a chave for outro objeto, o  
processo se repete

# Requisições

~~GET~~, POST, PUT, DELETE

## {JSON} Placeholder

- Consiste em um ambiente de API fake, disponibilizado apenas para testes de requisições de API.
- Ele permite a visualização de recursos padrão, o que pode nos ajudar a entender a dinâmica das requisições REST.
- O padrão de funcionamento aplicado nesse 'ambiente de testes' é o mesmo utilizado no consumo de qualquer outra API

# {JSON} Placeholder

Free fake API for testing and prototyping.

Powered by [JSON Server](#) + [LowDB](#). Tested with [XV](#)

Serving ~2 billion requests each month.

## When to use

JSONPlaceholder is a free online REST API that you can use **whenever you need some fake data**. It can be in a README on GitHub, for a demo on CodeSandbox, in code examples on Stack Overflow, ...or simply to test things locally.

## Resources

JSONPlaceholder comes with a set of 6 common resources:

<a href="#">/posts</a>	100 posts
<a href="#">/comments</a>	500 comments
<a href="#">/albums</a>	100 albums
<a href="#">/photos</a>	5000 photos
<a href="#">/todos</a>	200 todos
<a href="#">/users</a>	10 users

**Note:** resources have relations. For example: posts have many comments, albums have many photos, ... see [guide](#) for the full

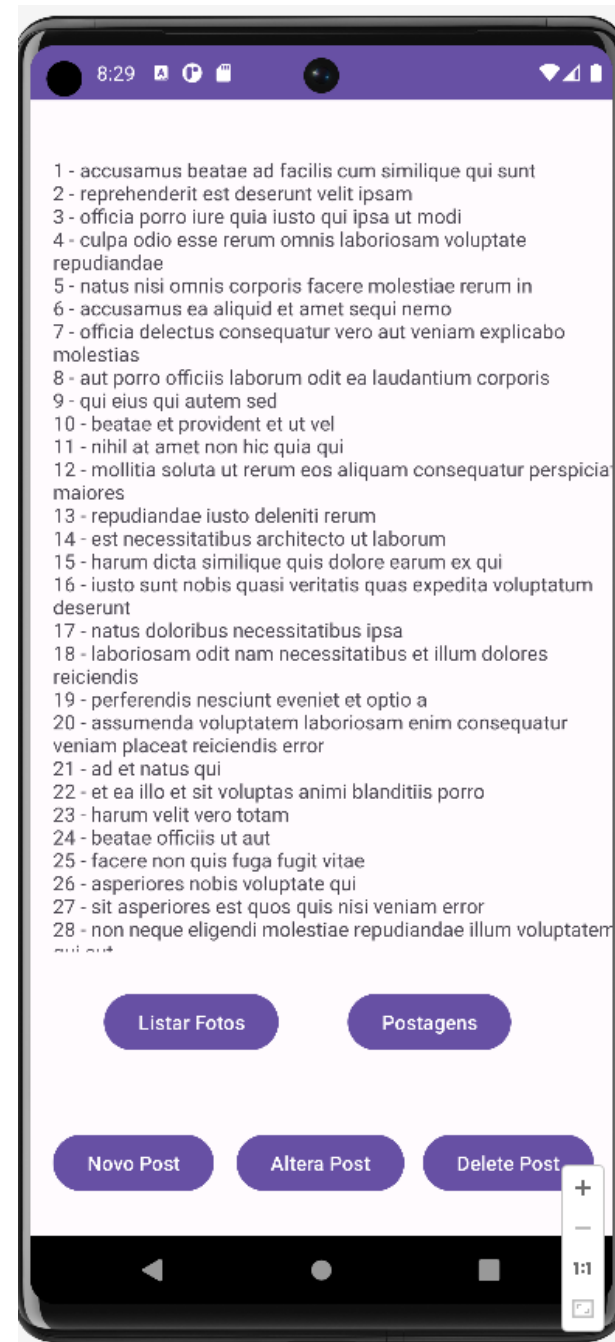
# Routes

All HTTP methods are supported. You can use http or https for your requests.

GET	<a href="#">/posts</a>
GET	<a href="#">/posts/1</a>
GET	<a href="#">/posts/1/comments</a>
GET	<a href="#">/comments?postId=1</a>
POST	<a href="#">/posts</a>
PUT	<a href="#">/posts/1</a>
PATCH	<a href="#">/posts/1</a>
DELETE	<a href="#">/posts/1</a>



**Note:** see [guide](#) for usage examples.



# Ambiente de testes

## {JSON} Placeholder



```
public class MainActivity extends AppCompatActivity {
```

6 usages

```
private TextView txtRetornaDados;
```

2 usages

```
private Button btGetFotos;
```

2 usages

```
private Button btGetPostagens;
```



```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    txtRetornaDados = findViewById(R.id.txtRetornaDados);
```

```
    btGetFotos = findViewById(R.id.btGetFotos);
```

```
    btGetPostagens = findViewById(R.id.btGetPostagens);
```

```
    //Configura o objeto retrofit.
```

```
    retrofit = new Retrofit.Builder()
```

```
        .baseUrl("https://jsonplaceholder.typicode.com/")
```

```
        .addConverterFactory(GsonConverterFactory.create())
```

```
        .build();
```

```
    btGetFotos.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
```

```
        public void onClick(View v) { RecuperFotos(); }
```

```
    });
```

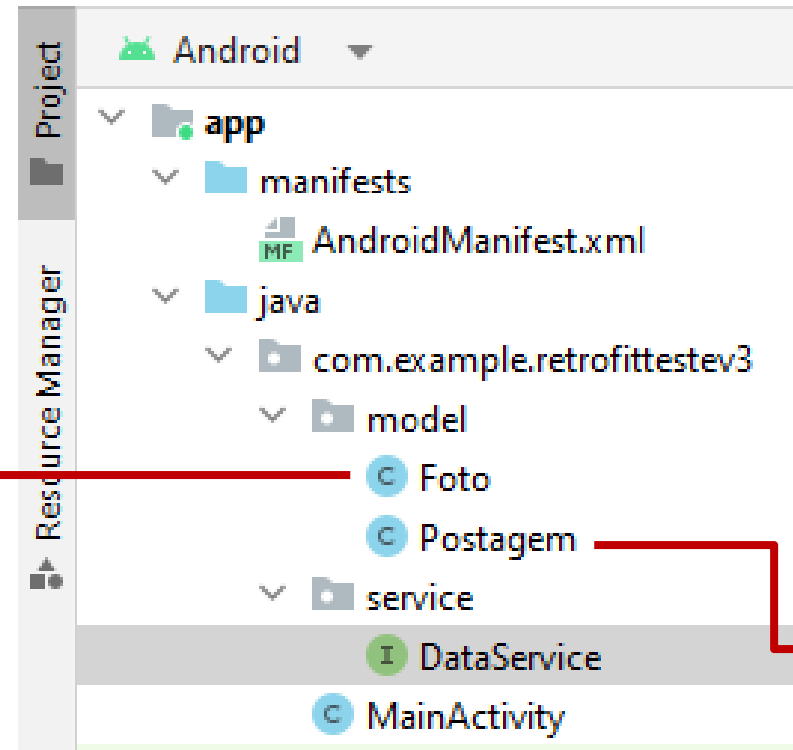
```
    btGetPostagens.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
```

```
        public void onClick(View v) { RecuperPostagens(); }
```

```
    });
```

```
public class Foto {  
    2 usages  
    private String albumId;  
    2 usages  
    private String id;  
    2 usages  
    private String title;  
    2 usages  
    private String url;  
    2 usages  
    private String thumbnailUrl;  
}
```



```
public class Postagem {  
    2 usages  
    private String userId;  
    2 usages  
    private String id;  
    2 usages  
    private String title;  
    2 usages  
    private String body;  
}
```

# Interface se Serviço

```
public interface DataService {
```

```
    1 usage
```

```
    @GET("/photos")
```

```
    Call<List<Foto>> RecuperaPhoto();
```

```
    1 usage
```

```
    @GET("/posts")
```

```
    Call<List<Postagem>> RecuperaPostagem();
```

```
    -
```



Utilizamos desta vez um **nome genérico para a interface**, de modo que possamos utilizá-la para acessar diferentes endpoints

```
private void RecuperFotos(){
    DataService service = retrofit.create(DataService.class);
    Call<List<Foto>> call = service.RecuperaPhoto();

    call.enqueue(new Callback<List<Foto>>() {
        @Override
        public void onResponse(Call<List<Foto>> call, Response<List<Foto>> response) {

            List<Foto> fotos = response.body();

            if (response.isSuccessful()) {
                String linha = "";
                for (Foto f: fotos) {
                    linha += "\n" + f.getId() + " - "
                        + f.getTitle();
                }
                txtRetornaDados.setText(linha);
            }
        }
    });
}
```



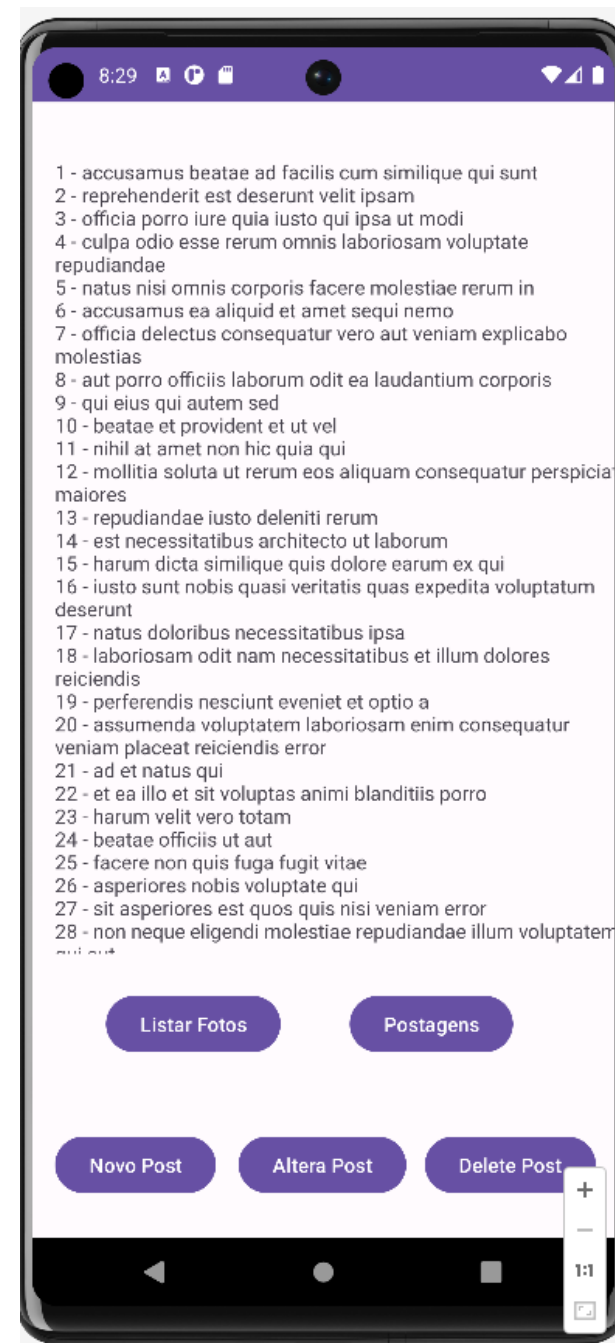
Observe que para esse endpoint é recebida uma lista de fotos. Para recuperar os dados, foi necessário iterar na lista recebida como payload

```
private void RecuperPostagens(){  
    DataService service = retrofit.create(DataService.class);  
    Call<List<Postagem>> call = service.RecuperaPostagem();  
  
    call.enqueue(new Callback<List<Postagem>>() {  
        @Override  
        public void onResponse(Call<List<Postagem>> call, Response<List<Postagem>> response) {  
  
            List<Postagem> posts = response.body();  
  
            if (response.isSuccessful()) {  
                String linha = "";  
                for (Postagem p: posts) {  
                    linha += "\n" + p.getId() + " - "  
                        + p.getTitle();  
                }  
                txtRetornaDados.setText(linha);  
            }  
        }  
    })  
    @Override  
    public void onFailure(Call<List<Postagem>> call, Throwable t) {}  
});
```

O método de **recuperação de postagens** funciona de forma similar, requerendo apenas alguns ajustes.

```
https://jsonplaceholder.typicode.com/photos

[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
  },
  {
    "albumId": 1,
    "id": 3,
    "title": "officia porro iure quia iusto qui ipsa ut modi",
    "url": "https://via.placeholder.com/600/24f355",
    "thumbnailUrl": "https://via.placeholder.com/150/24f355"
  },
  {
    "albumId": 1,
    "id": 4,
    "title": "culpa odio esse rerum omnis laboriosam voluptate repudiandae",
    "url": "https://via.placeholder.com/600/d32776",
    "thumbnailUrl": "https://via.placeholder.com/150/d32776"
  }
]
```



# Requisições POST

11 usages

```
public interface DataService {
```

1 usage

```
@GET("/photos")
```

```
Call<List<Foto>> RecuperaPhoto();
```

1 usage

```
@GET("/posts")
```

```
Call<List<Postagem>> RecuperaPostagem();
```



1 usage

```
@POST("/posts")
```

```
Call<Postagem> SalvarPostagem(@Body Postagem postagem);
```

As requisições do tipo post devem carregar um *payload*, que é embarcado na requisição HTTP

O atributo body indica o conjunto de dados a ser transmitido ao endpoint



2 usages

```
public class MainActivity extends AppCompatActivity {
```

6 usages

```
private TextView txtRetornaDados;
```

2 usages


```
private Button btGetFotos;
```

2 usages


```
private Button btGetPostagens;
```

2 usages

```
private Button btPostar;
```



```
    btPostar.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) { Postar(); }  
    });
```



```
private void Postar(){
```

```
    DataService service = retrofit.create(DataService.class);
```

```
    Postagem p = new Postagem();
    p.setId("12345");
    p.setUserId("54321");
    p.setTitle("Postagem de Teste");
    p.setBody("Conteúdo da postagem");
```

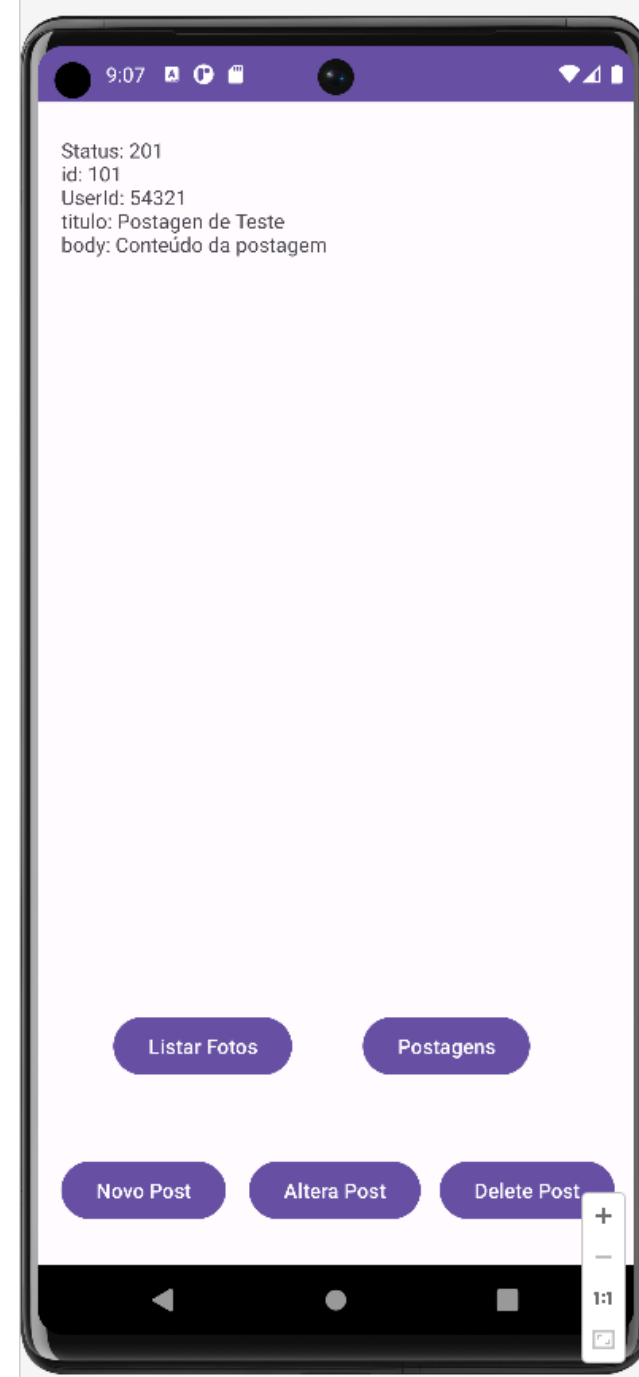
```
    Call<Postagem> call = service.SalvarPostagem(p);
```

```
    call.enqueue(new Callback<Postagem>() {
        @Override
        public void onResponse(Call<Postagem> call, Response<Postagem> response) {
```

```
            if (response.isSuccessful()) {
                Postagem postagem = response.body();
```

```
                txtRetornaDados.setText("Status: " + response.code() + "\n" +
                    "id: " + postagem.getId() + "\n" +
                    "UserId: " + postagem.getUserId() + "\n" +
                    "titulo: " + postagem.getTitle() + "\n" +
                    "body: " + postagem.getBody());
```

```
            }
        }
        @Override
        public void onFailure(Call<Postagem> call, Throwable t) {}
    });
}
```



# Requisições PUT

```
public interface DataService {}
```

```
1 usage
```

```
@GET("/photos")
```

```
Call<List<Foto>> RecuperaPhoto();
```

```
1 usage
```

```
@GET("/posts")
```

```
Call<List<Postagem>> RecuperaPostagem();
```

```
1 usage
```

```
@POST("/posts")
```

```
Call<Postagem> SalvarPostagem(@Body Postagem postagem);
```

```
1 usage
```

```
@PUT("/posts/{id}")
```

```
Call<Postagem> AlteraPostagem(@Path("id") int id, @Body Postagem postagem);
```



Uma requisição do tipo PUT pode enviar um parâmetro para facilitar a localização do recurso e um payload que contém o conjunto de dados a ser alterado

```
public class MainActivity extends AppCompatActivity {
```

6 usages

```
private TextView txtRetornaDados;
```

2 usages

```
private Button btGetFotos;
```

2 usages

```
private Button btGetPostagens;
```

2 usages

```
private Button btPostar;
```

2 usages

```
private Button btUpdatePost;
```



```
    btUpdatePost.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) { UpdatePostagem(); }  
    });
```

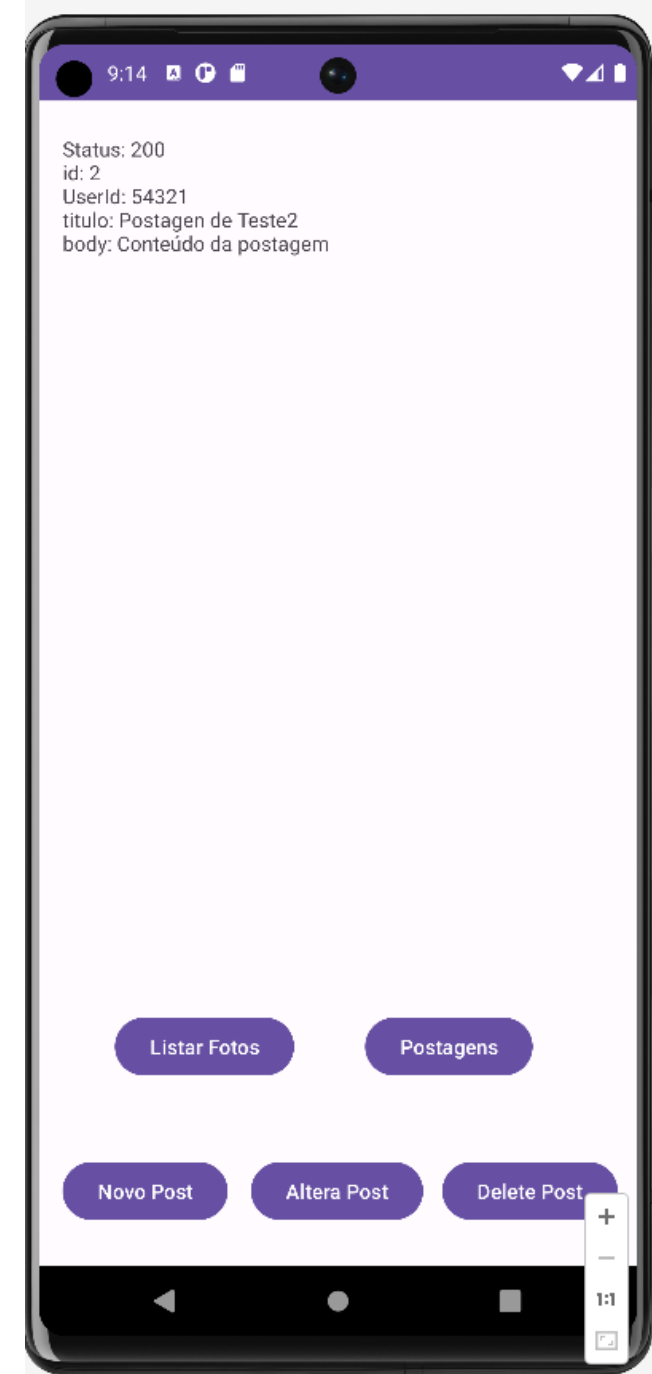
```
private void UpdatePostagem(){

    DataService service = retrofit.create(DataService.class);

    Postagem post = new Postagem();
    post.setId("12345");
    post.setUserId("54321");
    post.setTitle("Postagem de Teste2");
    post.setBody("Conteúdo da postagem");

    Call<Postagem> call = service.AlterarPostagem( id: 2,post);

    call.enqueue(new Callback<Postagem>() {
        @Override
        public void onResponse(Call<Postagem> call, Response<Postagem> response) {
            if (response.isSuccessful()) {
                Postagem postagem = response.body();
                txtRetornaDados.setText("Status: " + response.code() + "\n" +
                    "id: " + postagem.getId() + "\n" +
                    "UserId: " + postagem.getUserId() + "\n" +
                    "titulo: " + postagem.getTitle() + "\n" +
                    "body: " + postagem.getBody());
            }
        }
        @Override
        public void onFailure(Call<Postagem> call, Throwable t) {}
    });
}
```



Requisições DELETE

```
public interface DataService {}
```

1 usage

```
@GET("/photos")
```

```
Call<List<Foto>> RecuperaPhoto();
```

1 usage

```
@GET("/posts")
```

```
Call<List<Postagem>> RecuperaPostagem();
```

1 usage

```
@POST("/posts")
```

```
Call<Postagem> SalvarPostagem(@Body Postagem postagem);
```

1 usage

```
@PUT("/posts/{id}")
```

```
Call<Postagem> AlteraPostagem(@Path("id") int id, @Body Postagem postagem);
```

1 usage

```
@DELETE("/posts/{id}")
```

```
Call<Postagem> DeletePostagem(@Path("id") int id);
```

A requisição do tipo **DELETE** requerem apenas o ID do objeto a ser excluído.





2 usages

```
public class MainActivity extends AppCompatActivity {
```

6 usages

```
private TextView txtRetornaDados;
```

2 usages

```
private Button btGetFotos;
```

2 usages

```
private Button btGetPostagens;
```

2 usages


```
private Button btPostar;
```

2 usages


```
private Button btUpdatePost;
```

2 usages

```
private Button btDeletaPost;
```

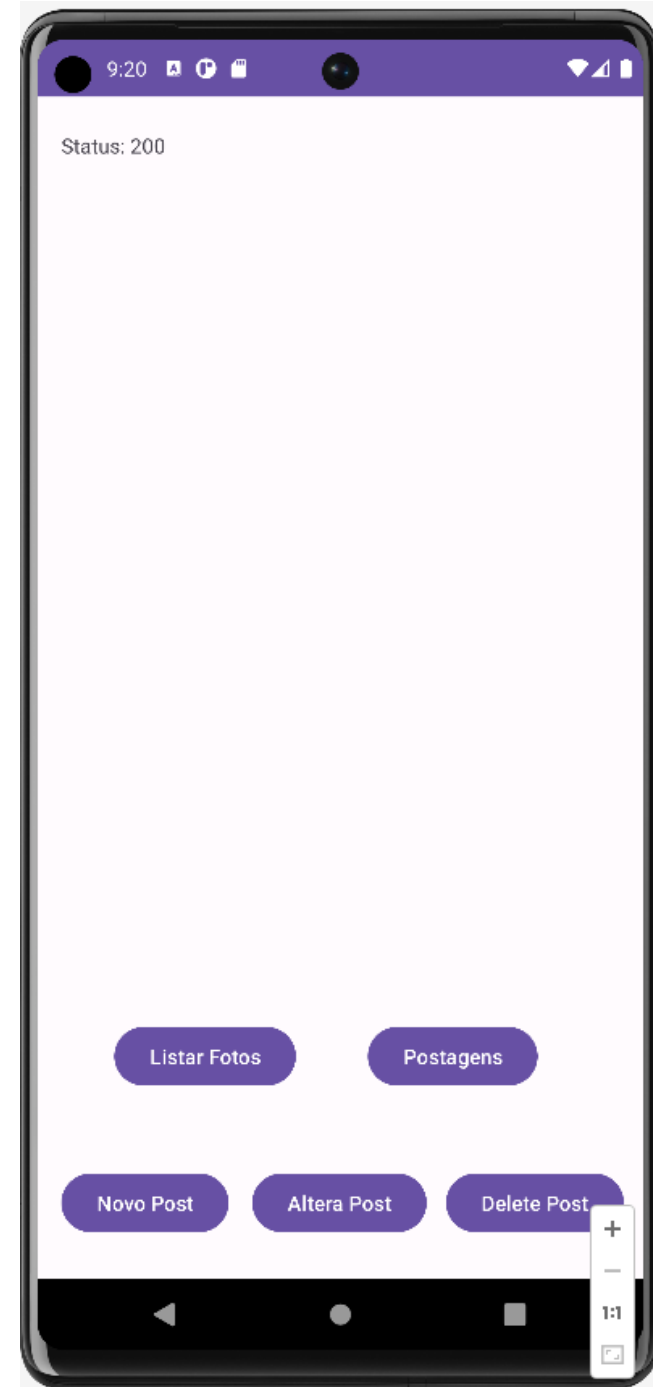


```
btDeletaPost.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) { DeletaPost(); }  
});
```



1 usage

```
private void DeletaPost(){  
  
    DataService service = retrofit.create(DataService.class);  
    Call<Postagem> call = service.DeletePostagem( id: 2);  
  
    call.enqueue(new Callback<Postagem>() {  
        @Override  
        public void onResponse(Call<Postagem> call, Response<Postagem> response) {  
            if (response.isSuccessful()) {  
                Postagem postagem = response.body();  
                txtRetornaDados.setText("Status: " + response.code());  
            }  
        }  
        @Override  
        public void onFailure(Call<Postagem> call, Throwable t) {}  
    });  
}
```



## Revisão

- Implemente o conversor de moedas usando a biblioteca Retrofit
- API: <https://docs.awesomeapi.com.br/api-de-moedas>

**Bons Estudos!**