

# Consumindo Serviços da Web

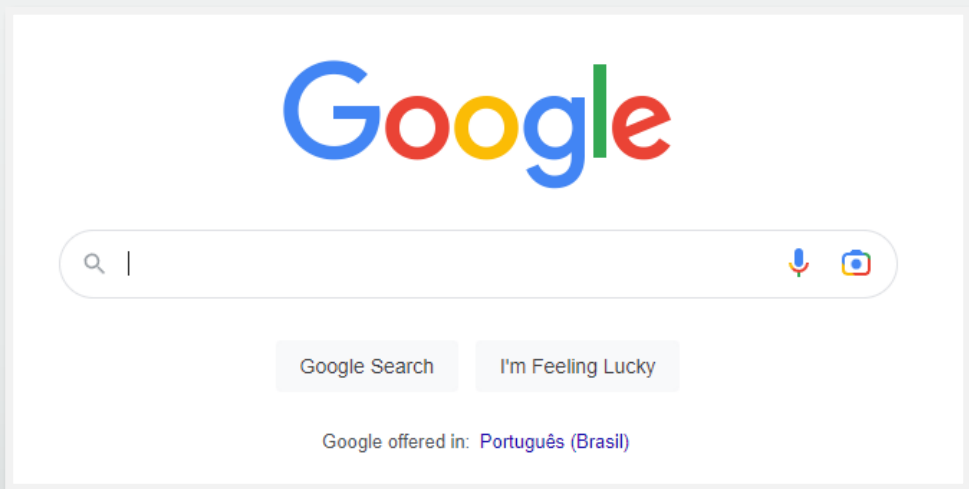
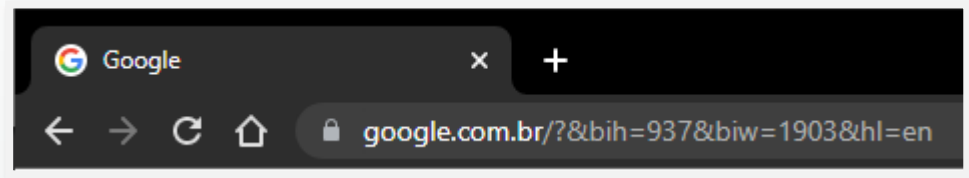
Dispositivos Móveis

Prof. Edson Mota, PhD, MSc, PMP

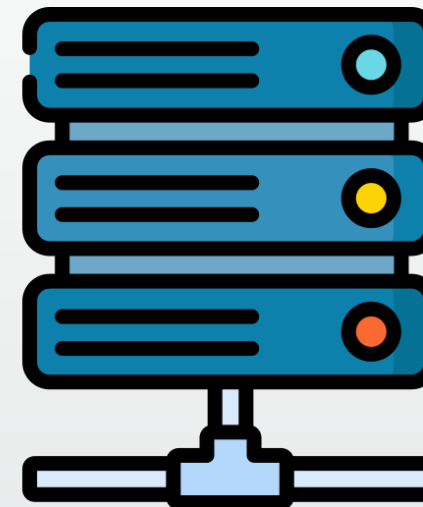
# Como funciona a comunicação via Internet?



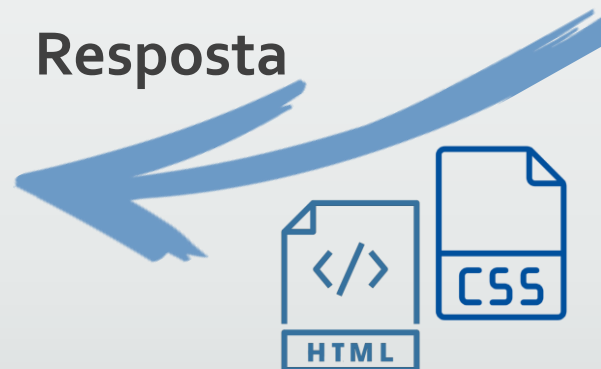
# Requisições Via Internet



Requisição

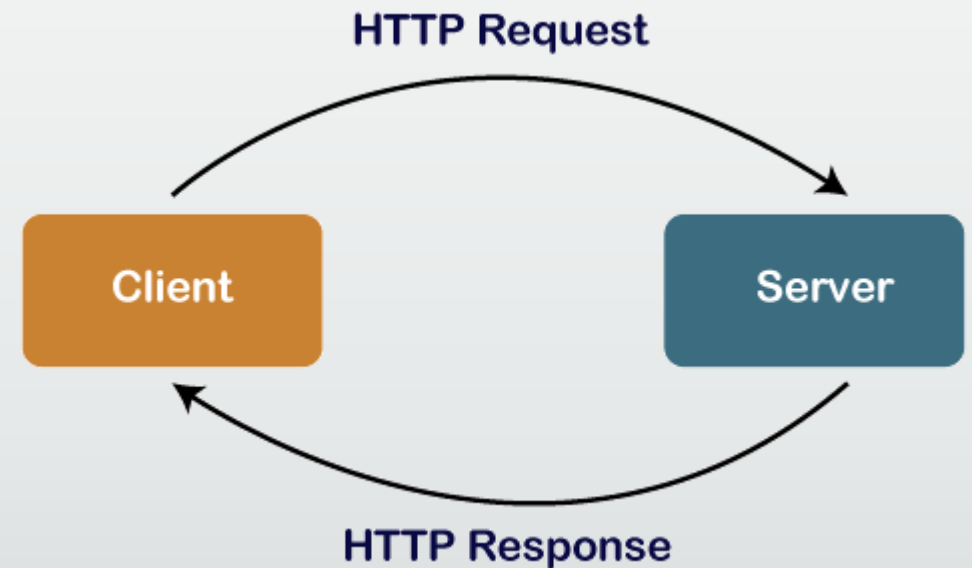


Resposta

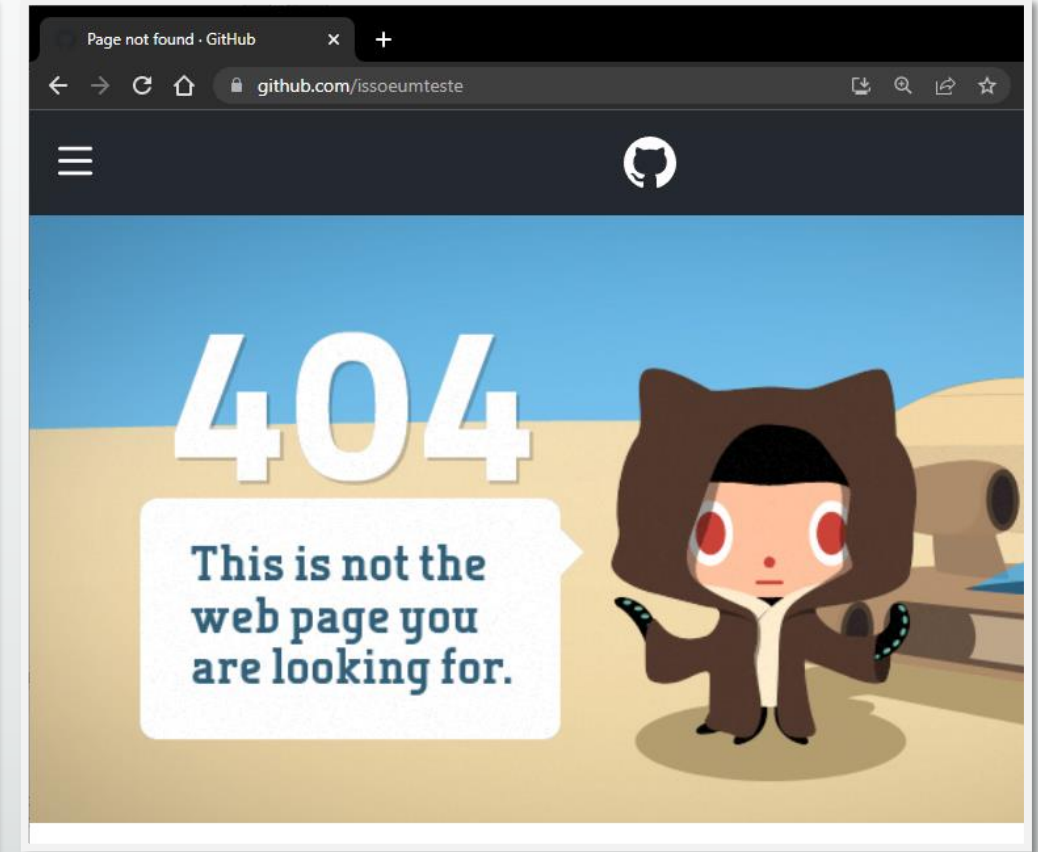
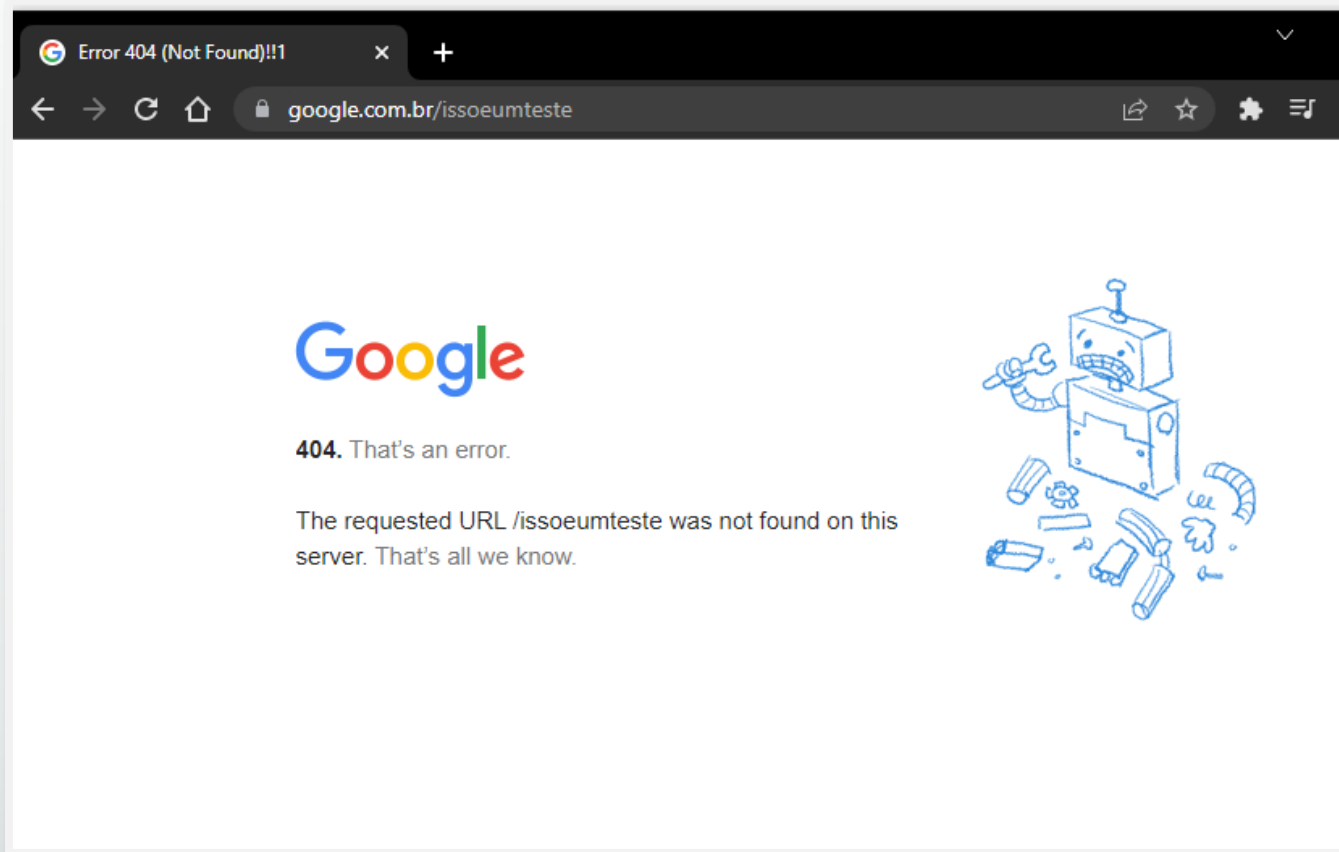


# Requisições HTTP

- HTTP (Hypertext Transfer Protocol)
  - Padronizou a forma como as mensagens são trocadas na web
- Tipos de requisições comuns:
  - Get → Recupera dados no servidor
  - Post → Cria dados no servidor
  - Put → Atualiza dados no servidor
  - Delete → Deleta dados no servidor



# Retornos – Códigos de Status

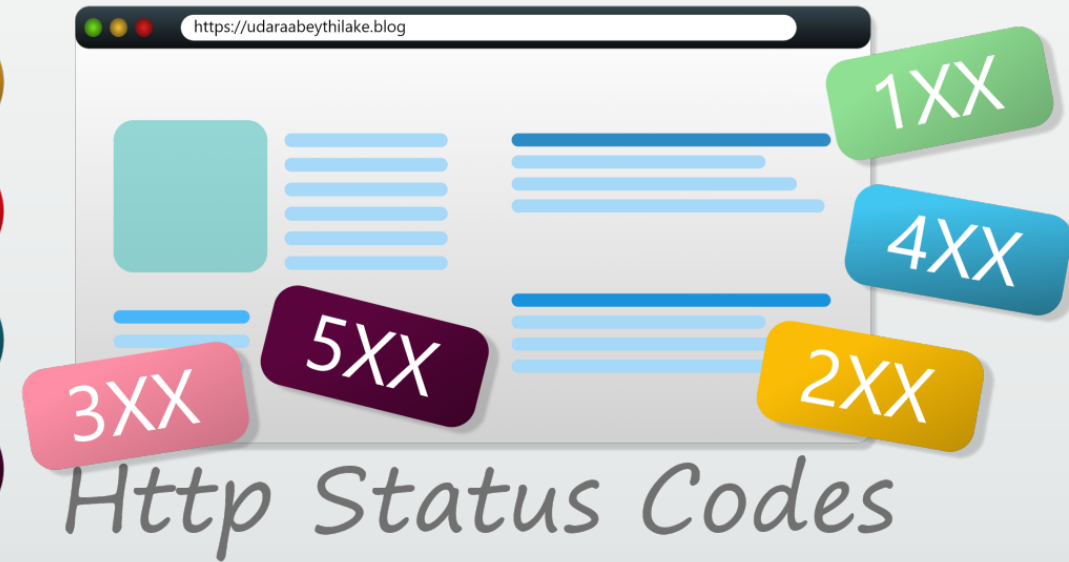


# Retornos – Códigos de Status

- Retornos mais conhecidos

- 404 → Not Found
- 200 → Ok
- 501 → Bad Gateway

- ... Existem muuuitos outros.

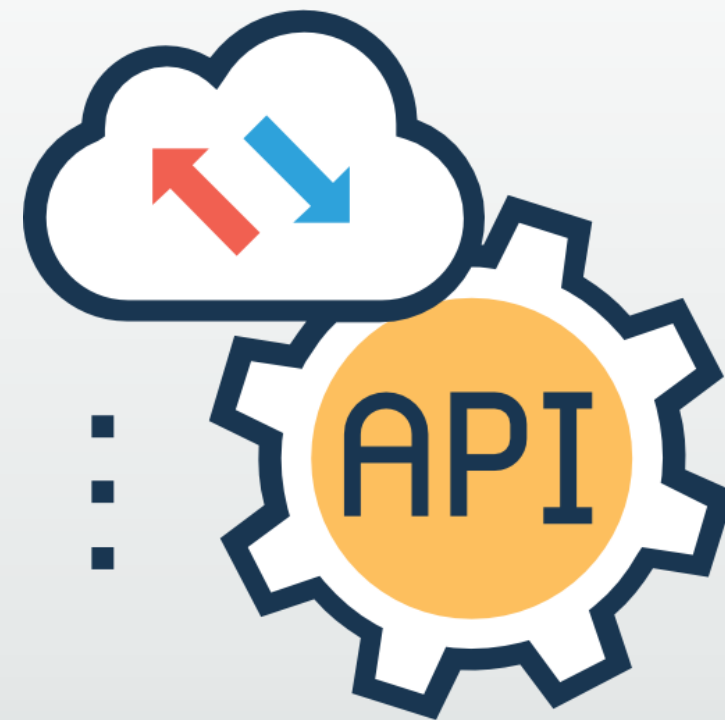


# O que é uma **API**?



# APIs

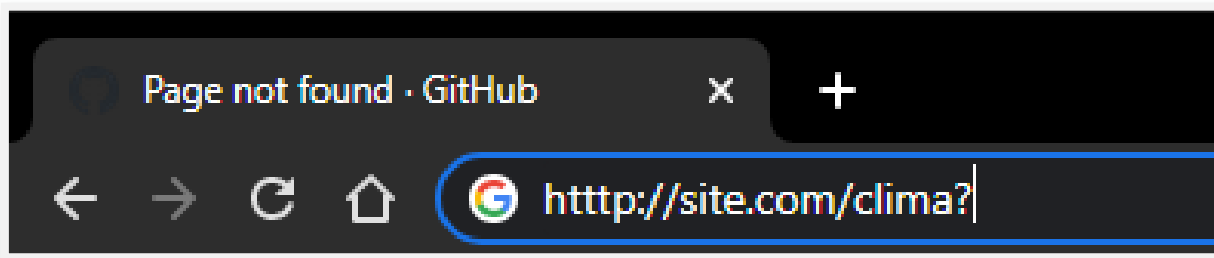
- Podemos entender APIs como um conjunto de rotinas e padrões de programação que permitem a criação de aplicativos e a integração de diferentes sistemas.
- Muitas APIs utilizam protocolos padrões da web, como HTTP/HTTPS, para permitir a comunicação entre as aplicações sem que seja necessário estruturar novos modelos de comunicação.





# Consumo de APIs

- O consumo de APIs consiste em utilizar serviços disponíveis na internet que oferecem informações sobre diferentes temas.
- Essas APIs funcionam como serviços e como tal, algumas são pagas e outras livres para consumo através da internet.
- As APIs cumprem um papel importante no desenvolvimento de aplicações móveis, uma vez que permitem a utilização de serviços que são processados fora do contexto do dispositivo.

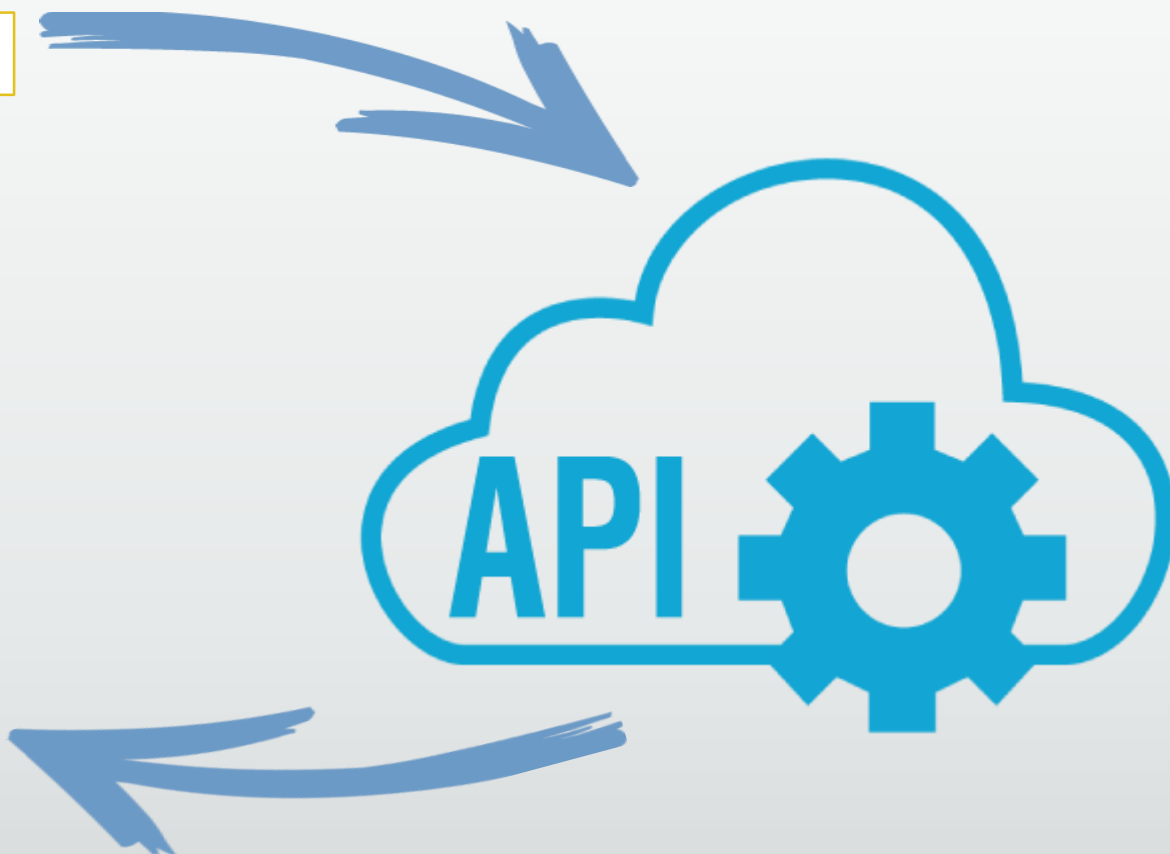


Muitas vezes ,as APIs requerem parâmetros , o símbolo “?” na URL indica a sintaxe para envio desses parâmetros.

http://site.com/**clima?**

cidade=ssa

```
{  
  "temperatura": 30,  
  "unidade": "Celsius"  
}
```



# Tipos de Retorno

- O retorno de uma API precisa ser independente da linguagem de programação que estamos utilizando.
- Os retornos mais utilizados são:
  - XML (Extensible Markup Language)
  - JSON (JavaScript Object Notation)

# Consumindo API



**END CONSUMERS**

**API**

**SOFTWARE  
APPLICATION**

# Localizando APIs

- Existem muitas APIs fornecendo os mais diversos tipos de dados.
- Em uma rápida consulta na internet (Google), podemos identificar diferentes tipos de serviços:
  - Serviço de CEP
  - Moedas e Conversões
  - Blockchain
  - Valor de ações
  - Redes sociais (Twitter, Facebook, Instagram, etc)
  - Serviço de Mapas
  - Entre outros

Exemplo: <https://viacep.com.br/>



ViaCEP - Webservice CEP e IBGE

viacep.com.br

# VIA CEP


Consulte CEPs de todo o Brasil

Procurando um [webservice](#) gratuito e de alto desempenho para consultar Códigos de Endereços? Utilize nosso serviço, melhore a qualidade de suas aplicações web e colabore para manter este projeto ativo.

## Acessando o webservice de CEP

Para acessar o webservice, um CEP no formato de **{8}** dígitos deve ser fornecido, por exemplo: 01001000. Após o CEP, deve ser fornecido o tipo de retorno desejado, que deve ser "json" ou "xml".

Exemplo de pesquisa por CEP:  
[viacep.com.br/ws/01001000/json/](https://viacep.com.br/ws/01001000/json/)







ViaCEP - Webservice CEP e IBGE


<https://viacep.com.br/ws/41650010/json/>

viacep.com.br/ws/41650010/json/

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

 Blockchain.com Início Preços Gráficos NFTs DeFi Academia Notícias Programadores Wallet Exchange Bitcoin Ethereum

Pesquisar Blockchain, Transações, Wallets e Blocos

 Sign In

# Exchange Rates API

## Market Prices and exchanges rates API

Some API calls are available with [CORS headers](#) if you add a &cors=true parameter to the GET request

URL: <https://blockchain.info/ticker>

No Parameters

Returns a JSON object with the currency codes as keys. "15m" is the 15 minutes delayed market price, "last" is the most recent market price, "symbol" is the currency symbol.

Response: 200 OK, application/json

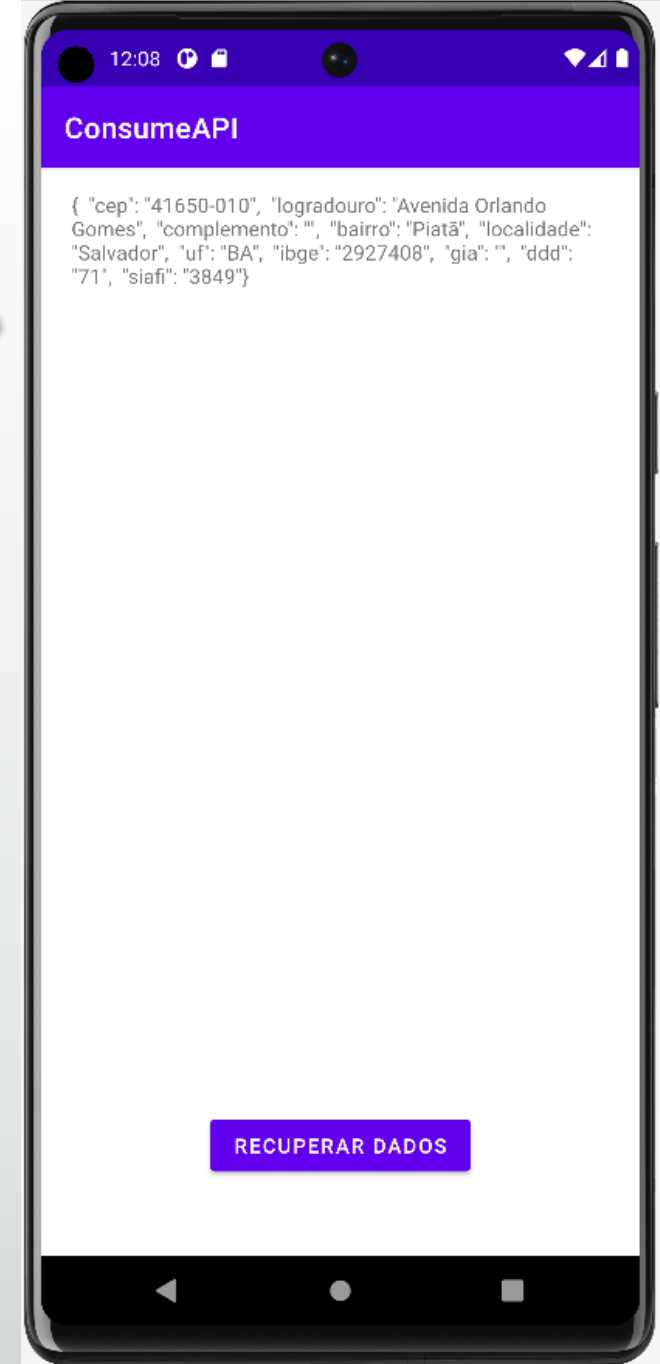
```
{
  "USD": {
    "15m": 57527.96,
    "last": 57527.96,
    "buy": 57527.96,
    "sell": 57527.96,
    "symbol": "$"
  },
  "AUD": {
    "15m": 74266.92,
    "last": 74266.92,
    "buy": 74266.92,
    "sell": 74266.92,
    "symbol": "$"
  },
  "BRL": {
    "15m": 308021.94,
    "last": 308021.94,
    "buy": 308021.94,
    "sell": 308021.94,
    "symbol": "R$"
  },
  "CAD": {
    "15m": 70575.82,
    "last": 70575.82,
    "buy": 70575.82,
    "sell": 70575.82,
    "symbol": "C$"
  }
}
```

[https://www.blockchain.com/pt/explorer/api/exchange\\_rates\\_api](https://www.blockchain.com/pt/explorer/api/exchange_rates_api)



A screenshot of a web browser window. The address bar shows the URL `https://viacep.com.br/ws/41650010/json/`. The page content displays a JSON object representing address data.

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



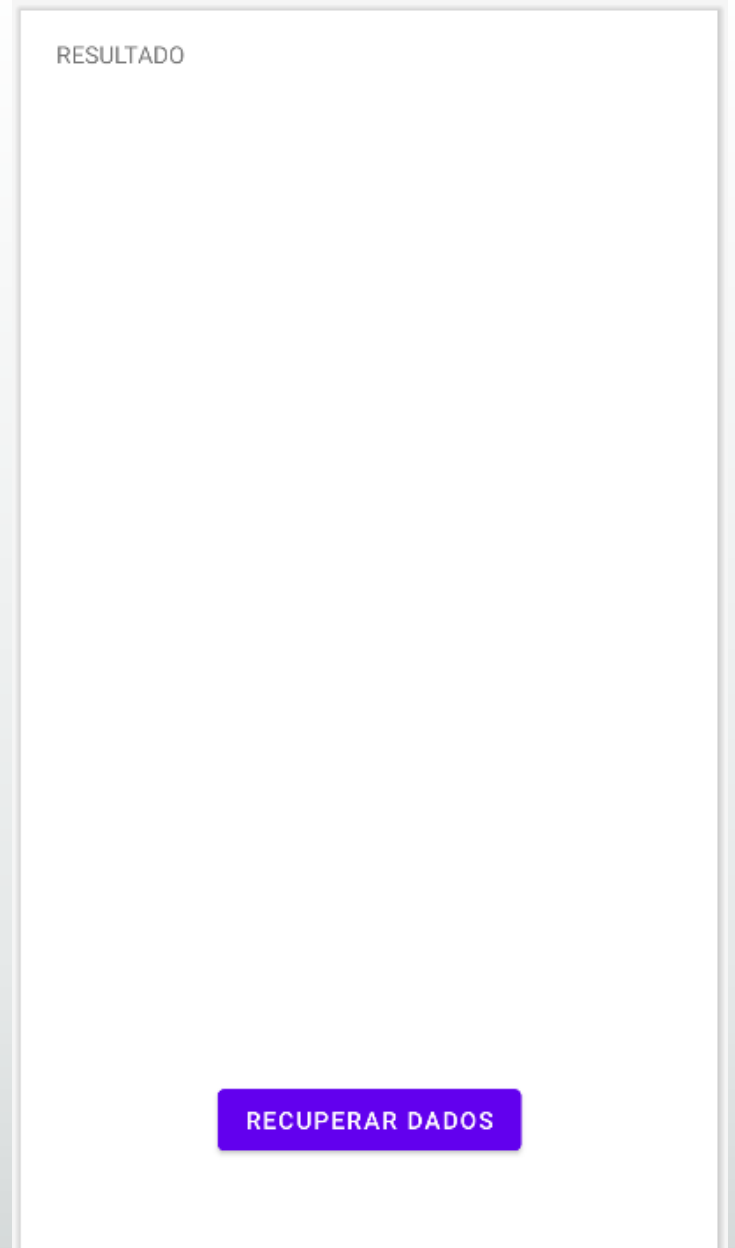


# Consumindo APIs

- Para este projeto, vamos precisar utilizar o conceito de `AsyncTask`
- Em APIs, as tarefas são essencialmente assíncronas, ou seja:
  - Uma requisição é enviada, mas não há a **espera** do retorno dessa mesma requisição.
  - Esses eventos ocorrem de forma desassociada (conceito de Thread ou processamento assíncrono).

# Criando Nosso APP

- Vamos começar criando a nossa activity.
- Nesse exemplo, utilize esse mesmo layout para facilitar o entendimento.
- O processamento esperado é:
  - Ao clicar em Recuperar Dados, a API deverá ser acionada e os dados trazidos ao TextView.



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

2 usages

```
private Button btRecuperaDados;
```

1 usage

```
private TextView txtResultado;
```

```
@Override
```

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

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

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

```
    btRecuperaDados = findViewById(R.id.btRecuperarDados);
```

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

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

```
        @Override
```

```
        public void onClick(View v) {
```

```
        }
```

```
    });
```

```
}
```



Vamos criar uma nova classe Assíncrona (dentro da classe MainActivity)

```
class MyTask extends AsyncTask|
```


- AsyncTask<Params, Progress, Result> android.os
- AsyncTaskLoader<D> androidx.loader.content
- AsyncTaskLoader<D> android.content
- AsynchronousCompletionTask java.util.concurrent.Complet...

Ctrl+Abaixo and Ctrl+Acima will move caret down and up in the editor [Next Tip](#)

- AsyncTask é uma classe abstrata em Java que permite executar operações de longa duração em segundo plano (background).
- A operação requer três parâmetros:
  1. Tipo de entrada para a tarefa (ex: A URL da API).
  2. Tipo de unidade de progresso (podemos configurar aqui um ProgressBar, por exemplo)
  3. Resultado a ser retornado pela tarefa após a conclusão

# Implementando MyTask

```
class MyTask extends AsyncTask<String, Void, String> {  
  
    @Override  
    protected String doInBackground(String... strings) {  
        return null;  
    }  
}
```



- Vamos sobrescrever mais dois métodos:
- OnPreExecute();
- OnPostExecute();

Implementação  
requerida

# Métodos da AsyncTask

- **onPreExecute ()** : é chamado **antes** da execução da tarefa em segundo plano.
- **doInBackground ()** : é o método principal do AsyncTask, onde a tarefa em segundo plano é executada.
- **onProgressUpdate ()** : é chamado durante a execução da tarefa em segundo plano para atualizar a interface do usuário com o progresso da tarefa.
- **onPostExecute ()** : é chamado após a conclusão da tarefa em segundo plano. Esse processamento é executado na thread principal (UI Thread).

# Instanciando MyTask / **doInBackground()**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    btRecuperaDados = findViewById(R.id.btRecuperarDados);
    txtResultado = findViewById(R.id.txtResultado);

    btRecuperaDados.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            MyTask task = new MyTask();
            String urlApi = "https://blockchain.info/ticker"; // acesso API blockchain.com
            task.execute(urlApi);
        }
    });
}
```

```
@Override
protected String doInBackground(String... strings) {
    return null;
}
```

# Implementando o **doInBackground**

Observe que esses comandos requerem tratamento de exceção

```
@Override
protected String doInBackground(String... strings) {

    String urlString = strings[0];

    try {

        URL url = new URL(urlString);
        HttpURLConnection conexao = (HttpURLConnection) url.openConnection();

        // Recuperando dados
        //.....

    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    return null;
}
```

Cria o objeto URL

Realiza o acesso à API retornando os dados de conexão

Trata erros de formação de URL

Trata erros que ocorram durante a recuperação dos dados da API em disco



```

@Override
protected String doInBackground(String... strings) {

    String urlString = strings[0];
    InputStream inputStream = null;

    try {

        URL url = new URL(urlString);
        HttpURLConnection conexao = (HttpURLConnection) url.openConnection();

        // Recuperando dados
        inputStream = conexao.getInputStream();


    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    return inputStream.toString();
}

@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    txtResultado.setText(s);
}

```

Permite que requisições  
saiam para a internet



```

activity_main.xml x MainActivity.java x AndroidManifest.xml x
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="ConsumindoAPI"
13    android:supportRtl="true"
14    android:theme="@style/Theme.ConsumindoAPI"
15    tools:targetApi="31">

```



Embora o conteúdo da API tenha sido retornado, ele precisa ser convertido para um formato passível de visualização.

Nesse momento, ele é apenas um array de bytes.

```

@Override
protected String doInBackground(String... strings) {

    String urlString = strings[0];
    InputStream inputStream = null;
    InputStreamReader inputStreamReader = null;

    try {

        URL url = new URL(stringUrl);
        HttpURLConnection conexao = (HttpURLConnection) url.openConnection();

        // Array de Bytes
        inputStream = conexao.getInputStream();

        // decodifica para caracteres
        inputStreamReader = new InputStreamReader(inputStream);

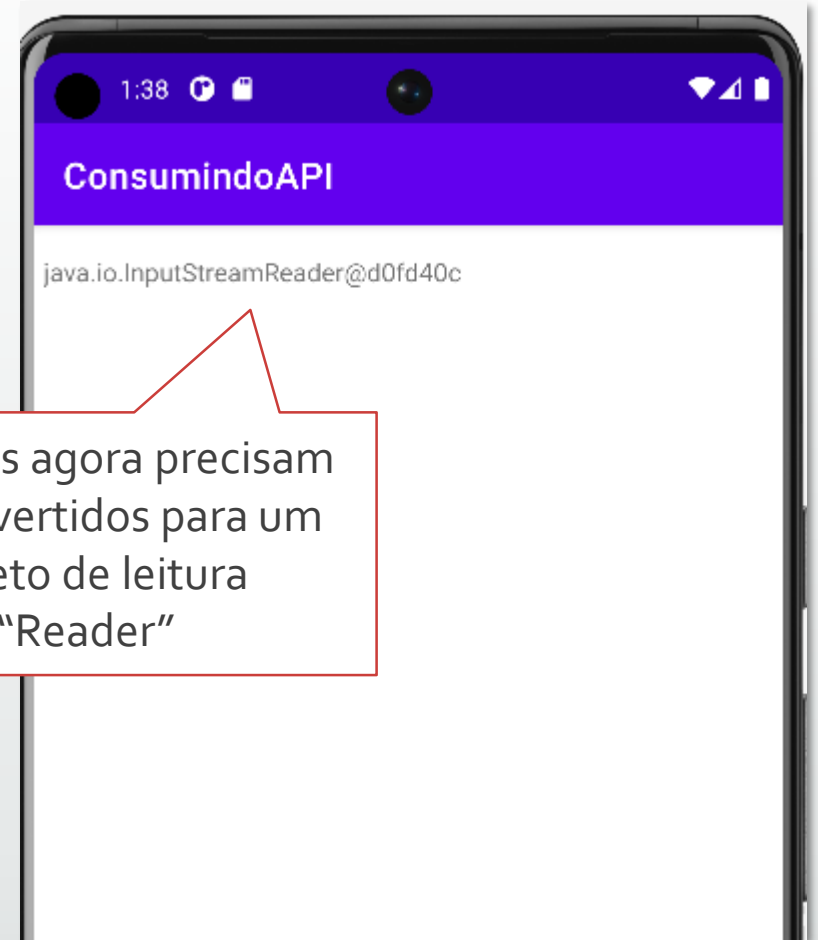
    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    return inputStreamReader.toString();
}

@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    txtResultado.setText(s);
}

```

Os dados agora precisam  
ser convertidos para um  
objeto de leitura  
"Reader"



```

@Override
protected String doInBackground(String... strings) {

    String urlString = strings[0];
    InputStream inputStream = null;
    InputStreamReader inputStreamReader = null;
    BufferedReader reader = null;
    StringBuffer buffer;

    try {

        URL url = new URL(stringUrl);
        HttpURLConnection conexao = (HttpURLConnection) url.openConnection();

        // Array de Bytes
        inputStream = conexao.getInputStream();

        // decodifica para caracteres
        inputStreamReader = new InputStreamReader(inputStream);

        //Leitura dos caracteres do inputStreamReader
        reader = new BufferedReader(inputStreamReader);

        buffer = new StringBuffer();

        String linha = ""; // armazena linha

        // faz a atribuição e testa se é nulo
        while ((linha=reader.readLine()) != null) {
            buffer.append(linha);
        }

    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    return buffer.toString();
}

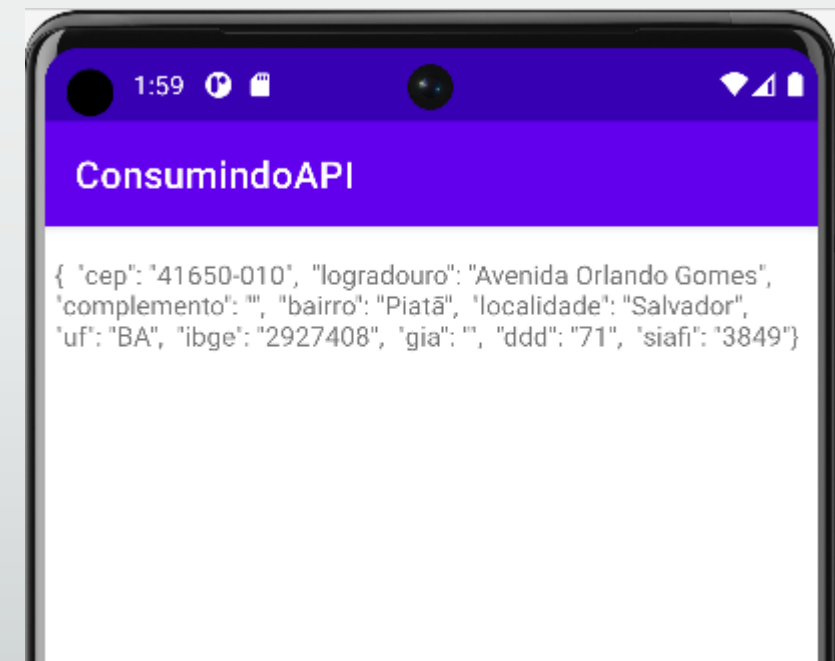
```



# E se alterarmos a **API**?



```
btRecuperaDados.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        MyTask task = new MyTask();  
        //String urlApi = "https://blockchain.info/ticker"; // acesso API blockchain.com  
        String urlApi = "https://viacep.com.br/ws/41650010/json/"; // ViaCEP  
        task.execute(urlApi);  
    }  
});
```



# JSON Files

# Objetos JSON (<https://www.json.org/json-pt.html>)

- Notação de Objetos JavaScript (JavaScript Object Notation)
- Trata-se de uma formatação leve para troca de dados que é comumente usada para intercâmbio de dados entre aplicativos web e serviços.
- A notação tornou-se tão popular porque:
  - Para seres humano é fácil de ler e escrever
  - Para máquinas é fácil de interpretar
  - Baseado em um subconjunto da linguagem JavaScript
  - Independente de linguagens de programação (formato texto)

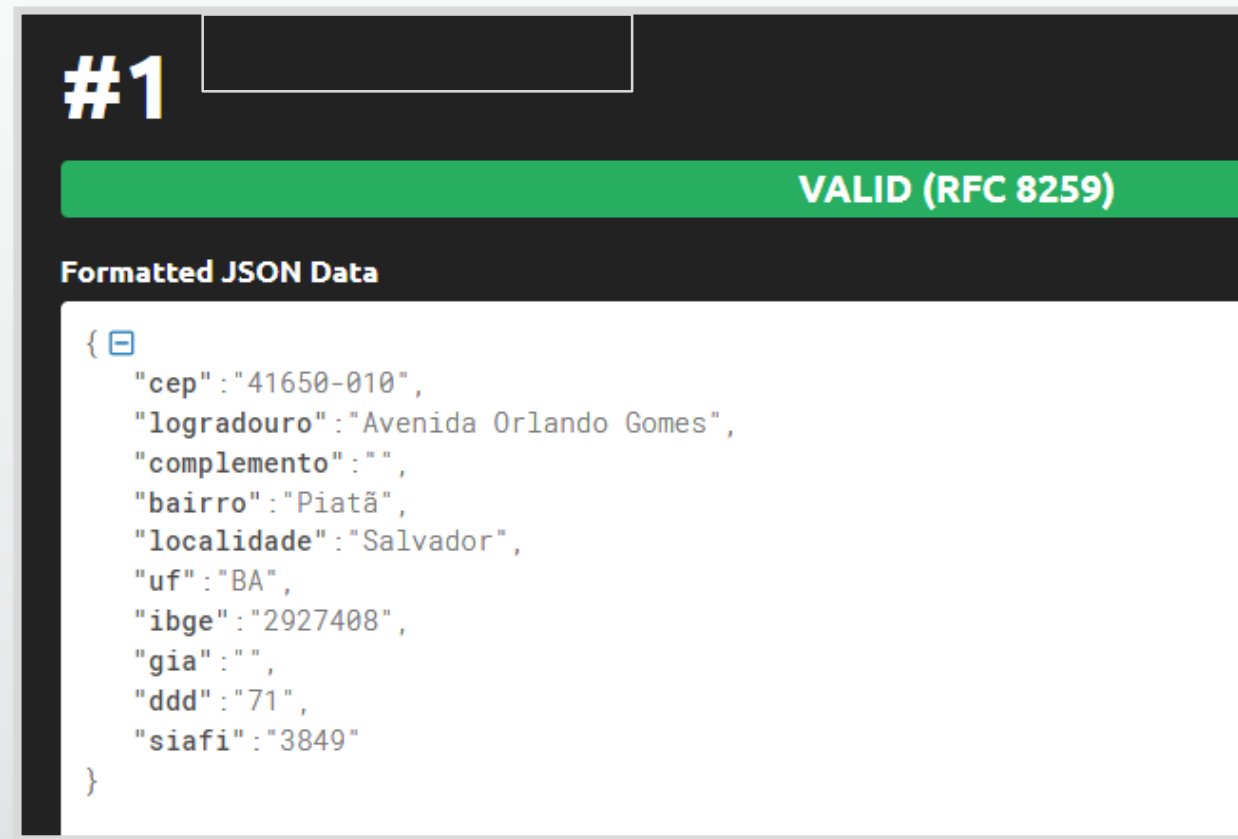


# Objetos JSON (<https://www.json.org/json-pt.html>)

- Está constituído de duas estruturas:
  1. Uma coleção de pares Chave/Valor
    - Essa estrutura alinha-se com conceitos de programação como Objetos, structs, dicionários, hash, entre outros.
  2. Uma lista de valores
    - Para muitas linguagens essas coleções são interpretadas como: Arrays, listas, vetores, etc.

# Validando JSONs

- Um arquivo JSON deve conter uma estrutura padronizada a partir de um conjunto de regras fundamentais.
- Uma ferramenta que pode ajudar nesse processo de formatação são os sites para validação de estruturas JSON.



<https://jsonformatter.curiousconcept.com/#>

# Estrutura Básica

- Em essência, trata-se de um conjunto de chave/valor, no qual o valor não necessariamente será composto de um único item.




## Formatted JSON Data

```
{
  "Nome": "Jsoe Souza",
  "CPF": "785458548-585",
  "Telefones": {
    "Comercial": "2565256-4564",
    "Residencial": "9585858585"
  }
}
```

# Processando Objetos JSON Retornados API

# Implementando o **onPostExecute**

@Override

```
protected void onPostExecute(String resultado) {  
    super.onPostExecute(resultado);  
  
    String logradouro = null;   
  
    try {  
        JSONObject jsonObjet = new JSONObject((resultado));   
        logradouro = jsonObjet.getString( name: "logradouro");   
        txtResultado.setText(logradouro);   
    } catch (JSONException e) {  
        throw new RuntimeException(e);  
    }  
}
```

Observe que é necessário passar a chave do JSON para que o objeto JSONObject seja capaz de retornar o valor.

@Override

```
protected void onPostExecute(String resultado) {
```

```
    super.onPostExecute(resultado);
```

```
    String cep, logradouro, complemento, bairro, localidade, uf, ddd = null;
```

```
    try {
```

```
        JSONObject jsonObjet = new JSONObject((resultado));
```

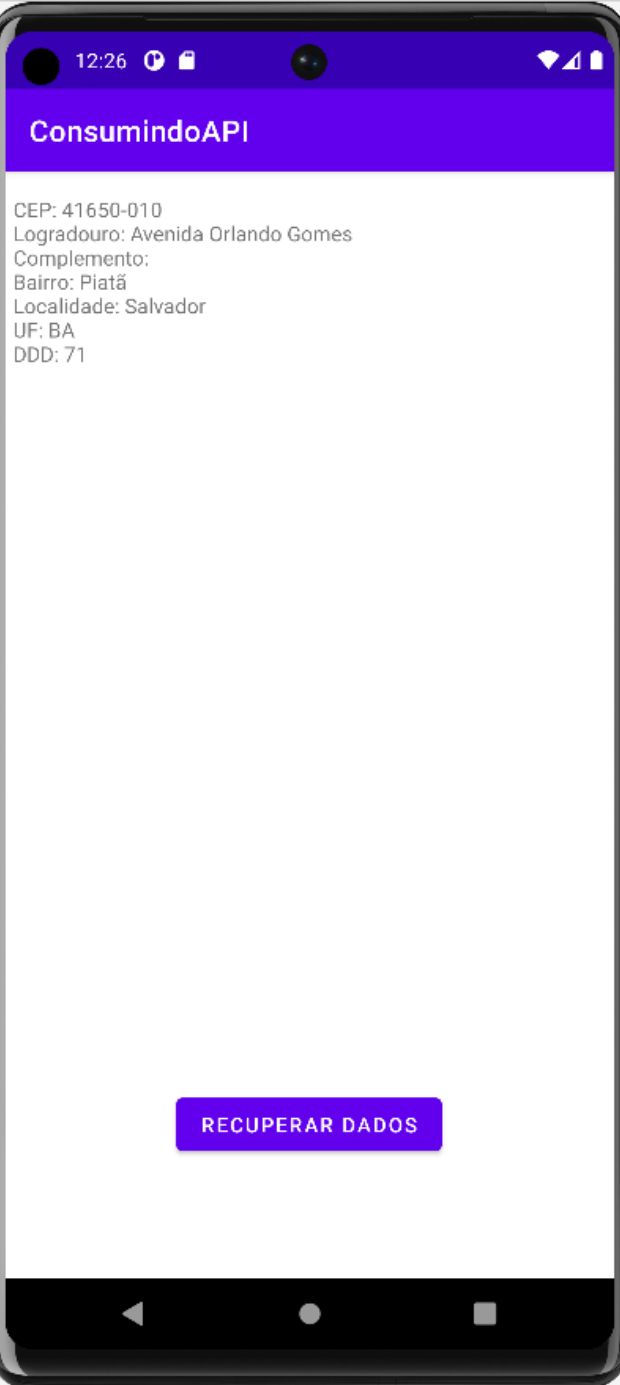
```
        cep = jsonObjet.getString( name: "cep");  
        logradouro = jsonObjet.getString( name: "logradouro");  
        complemento = jsonObjet.getString( name: "complemento");  
        bairro = jsonObjet.getString( name: "bairro");  
        localidade = jsonObjet.getString( name: "localidade");  
        uf = jsonObjet.getString( name: "uf");  
        ddd = jsonObjet.getString( name: "ddd");
```

```
        String txt = null;
```

```
        txt = String.format("CEP: %s\nLogradouro: %s" +  
                            "\nComplemento: %s" +  
                            "\nBairro: %s" +  
                            "\nLocalidade: %s" +  
                            "\nUF: %s" +  
                            "\nDDD: %s"  
                            , cep, logradouro, complemento, bairro, localidade, uf, ddd  
        );
```

```
        txtResultado.setText(txt);
```

```
    } catch (JSONException e) {  
        throw new RuntimeException(e);  
    }
```



**Bons Estudos!**