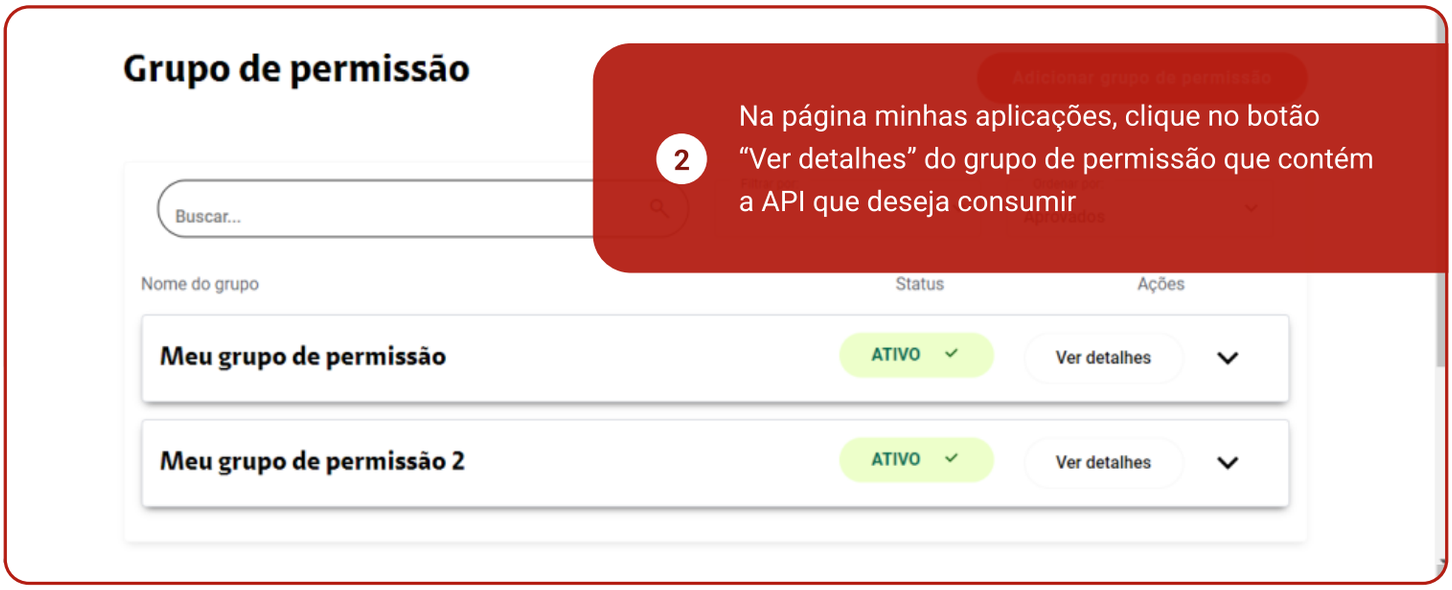
# Tutorial de consumo da API Face Match

Claro

Para iniciar o consumo da **API Face Match**, é necessário adquirir o token de acesso através da chave pública e privada da aplicação contratada no portal.

## » Como gerar um token na Base64

1 No menu à direita, clique em Minhas Aplicações



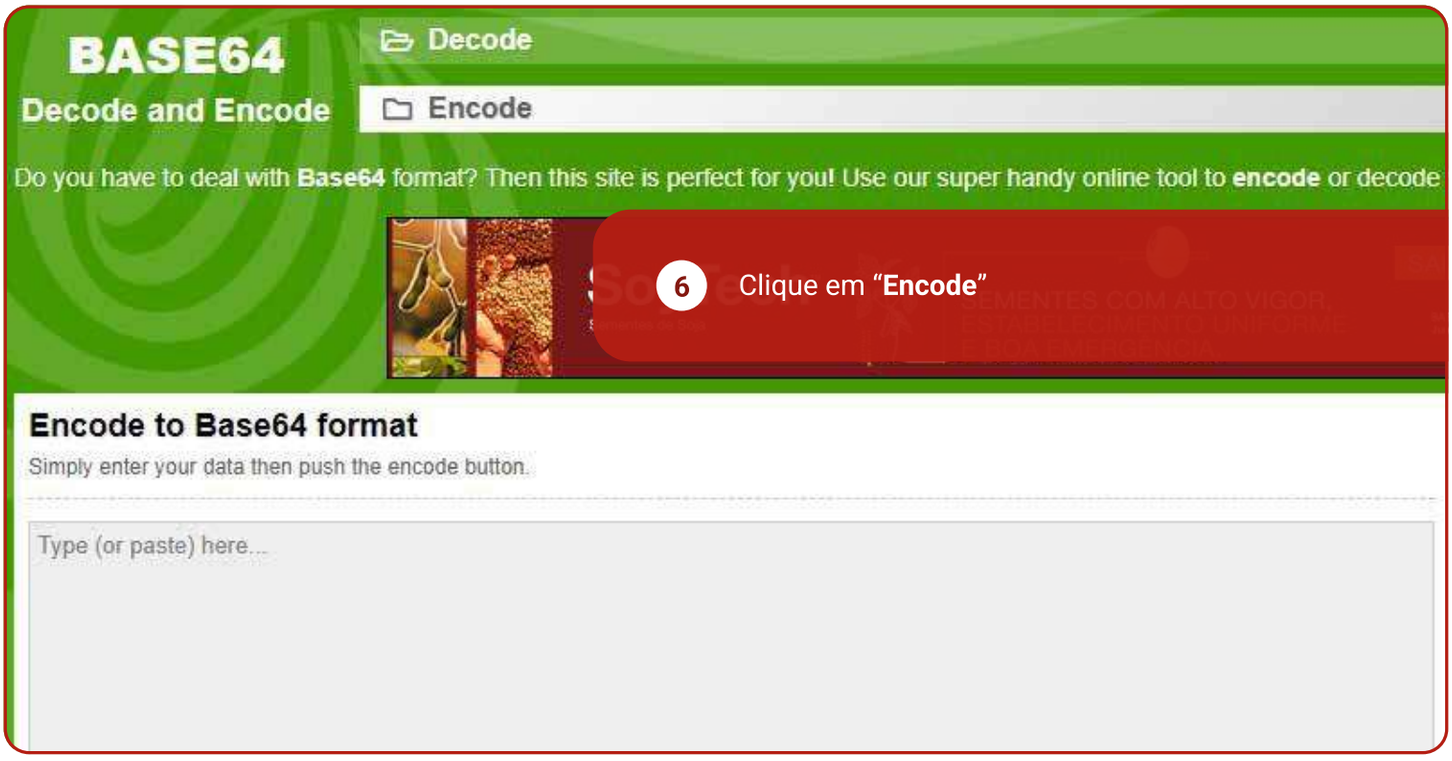

2 Na página minhas aplicações, clique no botão "Ver detalhes" do grupo de permissão que contém a API que deseja consumir

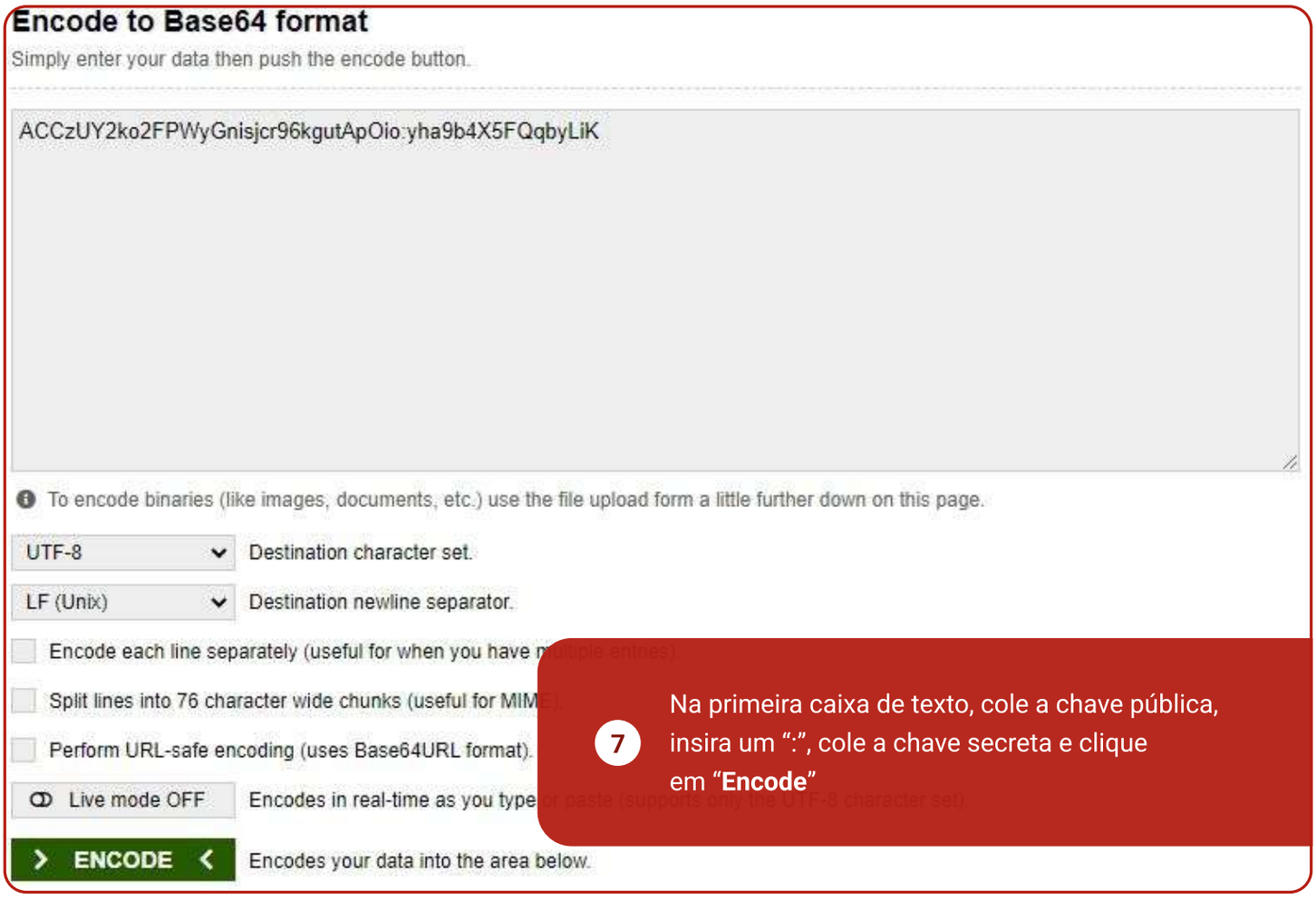
3 Clique na seta para ver os detalhes da aplicação desejada

4 Visualize as chaves pública e secreta, clique em "**Baixar arquivos**" e guarde a collection para utilizar mais tarde

Analytics

5 Acesse o site https://www.base64encode.org/.




**Ex:** ACCzUY2ko2FPWyGnisjcr96kgutApOio : yha9b4X5FQqbyLiK

**chave pública** : **chave secreta**

ENCODE Encodes your data into the area below.

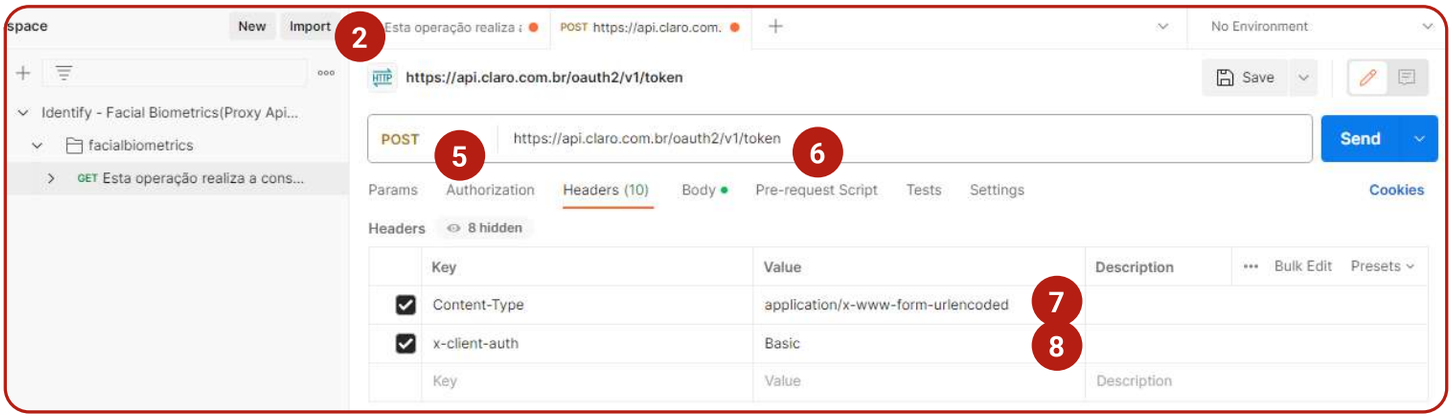
QUNDelVZMmtvMkZQV3lHbmlzamNyOTZrZ3V0QXBPaW86eWhhOWI0WDVGUXFieUxpSw==

8 Copie e guarde o token gerado para utilizar a seguir

O token não tem prazo de validade e pode ser utilizado por um período indeterminado, contanto que o status da aplicação permaneça **aprovado**

## Usar o postman para obter o access token da API Face Match

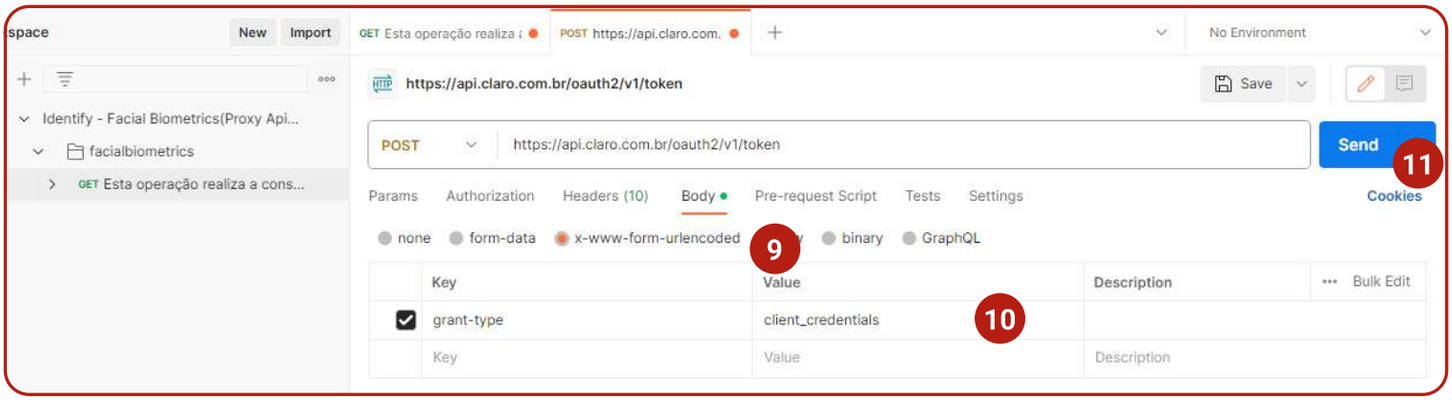
1. Inicie o postman
2. Clique no botão **Importar**
3. Selecione a **collection** baixada no passo anterior
4. Adicione uma **nova request** na collection
5. Selecione o método **POST**
6. Para a URL, insira **https://api.claro.com.br/oauth2/v1/token**
7. Na guia Headers, adicione chave de **Content-Type** e para o valor **application/x-www-form-urlencoded**
8. Na guia Headers, adicione a chave **x-client-auth** colando o valor gerado na **base64** após o texto Basic




9 Na guia **Body**, marque a caixa **x-www-form-urlencoded**

10 Na guia **Body**, adicione a chave **grant-type** com o valor **client_credentials**

11 Envie a solicitação pressionando o botão **Envio**




12 Salve o **access token** para consumir a API

> O **access token** gerado tem 24 horas de validade e deve ser solicitado novamente após esse período

# » Usar o postman para consumir a API Face Match

1 Na **collection** da API, clique na opção **Facial Biometrics**

2 Certifique-se que o método **GET** esteja selecionado

3 A URL inserida deverá ser **https://api.claro.com.br/identity/v1/facialbiometrics**

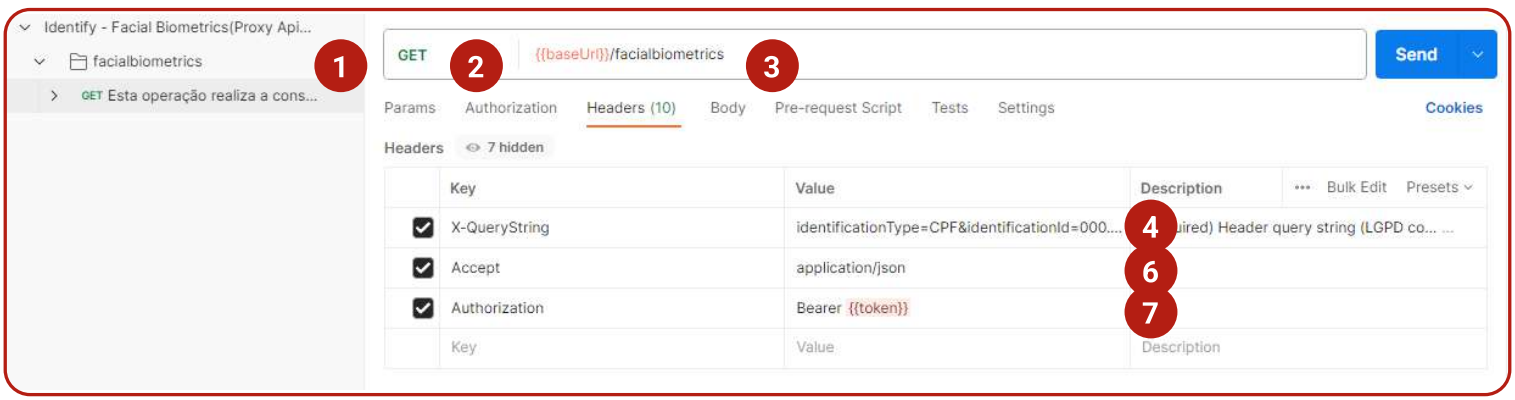
4 A guia **Headers** deve conter a chave **X-QueryString** com o valor da query string, onde são informados os campos **identificationType=valor&identificationId=valor**

5 O campo **identificationType**, obrigatório, tipo string, representa o **número de CPF**
O campo **identificationId**, obrigatório, tipo string, representa o **número do documento da pessoa consultada**

Os campos são do tipo string e precisam ser totalmente preenchidos.
Caso seja um CPF com menos de 11 caracteres deve ser sempre completado com zeros à esquerda.

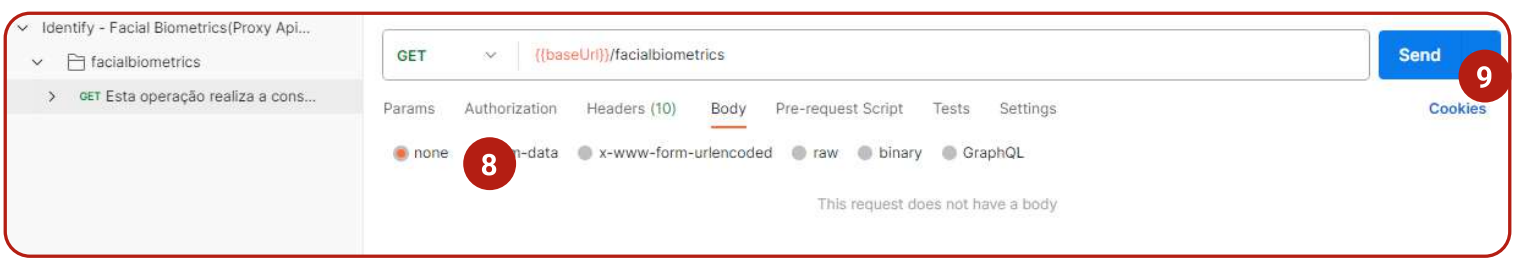
6 A guia **Headers** deve conter a chave **Accept** com o valor **application/json**

7 Na guia Headers, adicione chave **Authorization** e o valor **Bearer** com o access_token adquirido na solicitação do token




8 Na guia **Body**, marque a caixa **none**

9 Envie a solicitação pressionando o botão **Envio**




# » Dados de retorno com sucesso e erro

## Status de retorno com sucesso

Exemplo de json de retorno:

```
{
    "apiVersion": "<string>",
    "transactionId": "<string>",
    "data": {
        "exists": "<boolean>",
        "message": "<string>",
        "identificationType": "<string>",
        "identificationId": "<string>",
        "versions": [
            "<string>",
            "<string>"
        ]
        "biohashs": [
            "<string>",
            "<string>"
        ]
    }
}
```

## Explicação dos dados do json

1. Campo **apiVersion**, tipo string, representa a versão da API
2. Campo **transactionId**, tipo string, representa o UUID da transação
3. Campo **exists**, tipo boolean, indica se o CPF foi encontrado na base
4. Campo **message**, tipo string, representa a mensagem de retorno do backend
5. Campo **identificationType**, tipo string, representa o tipo de documento consultado
6. Campo **identificationId**, tipo string (11 caracteres), representa o número do documento da pessoa consultada
7. Campo **versions**, tipo string, representa a versão da API do backend
8. Campo **biohashs**, tipo string, representa a lista de hashs da biometria facial da pessoa consultada

## Status de retorno com erro

AJUSTAR DAQYU

Exemplo de json de retorno:

```
{
    "apiVersion": “<string>”,
    "transactionId": “<string>”,
    "error": {
        "httpCode": "<string>",
        "errorCode": "<string>",
        "message": "<string>",
        "detailedMessage": "<string>",
        "link": {
            "rel": "related",
            "href": "<string>"
        }
    }
}
```

## Explicação dos dados do json

1. Campo **apiVersion**, tipo string, representa a versão da API
2. Campo **transactionId**, tipo string, representa o UUID da transação
3. Campo **httpCode**, tipo string, representa o código de retorno HTTP
4. Campo **errorCode**, tipo string, representa o código de erro Claro
5. Campo **message**, tipo string, representa a Mensagem de erro
6. Campo **detailedMessage**, tipo string, representa a Mensagem detalhada
7. Campo **rel**, tipo string, sempre “related”
8. Campo **href**, tipo string, representa o endereço para o detalhamento dos erros

# » Status de response

## Código de retorno 200 (Success)

O usuário está com a conexão estabelecida na rede da operadora

## Código de retorno 400 (Bad Request)

A solicitação não pôde ser entendida pelo servidor devido à sintaxe incorreta. O cliente NÃO DEVE repetir o pedido sem modificações

## Código de retorno 401 (Unauthorized)

A solicitação requer autenticação do usuário

### Código de retorno 403 (Forbidden)

O servidor entendeu a solicitação, mas está se recusando a atendê-la

### Código de retorno 404 (Not Found)

O servidor não encontrou nada que corresponda ao request-URI (CPF não encontrado na base)

### Código de retorno 405 (Method Not Allowed)

O método solicitado não é suportado pela API

### Código de retorno 406 (Not Acceptable)

Servidor não pode produzir uma resposta que combine com a lista de valores aceitáveis definidos

### Código de retorno 414 (Request-URI Too Long)

O tamanho da URI requisitada é maior do que o tamanho que o servidor aceita interpretar

### Código de retorno 422 (Unprocessable Entity)

Caso seja retornado o errorCode “API-FACIALBIOMETRICS-001” significa que o cliente não foi encontrado na base de dados

### Código de retorno 429 (Too Many Requests)

O usuário enviou muitas solicitações em um determinado período de tempo

### Código de retorno 500 (Internal Server Error)

Erro interno do servidor

### Código de retorno 503 (Service Unavailable)

O servidor está indisponível no momento devido uma sobrecarga temporária ou manutenção programada

## » Exemplo de código em curl

```
!/bin/bash

url_facialbiometrics=https://api.claro.com.br/identity/v1/facialbiometrics
url_token=https://api.claro.com.br/oauth2/v1/token

identificationType=CPF
identificationId="98676489009"
key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
secret=XXXXXXXXXXXXX

senha=$( echo -n $key:$secret | base64)

curl -X POST $url_token \
-H 'Content-Type: application/x-www-form-urlenconded' \
-H 'cache-control: no-cache' \
-d 'grant_type=client_credentials' \
-H "x-client-auth: Basic $senha" > json
grep 'acess_token' json > atributoJson \

token=$ (cut -d' " ' -f4 atributoJson )

curl -X  'GET ' $url_facialbiometrics \
-H 'Content-Type: application/json' \
-H  'X-QueryString:
identificationType=' "$identificationType" '&IdentificationId=' "$identificationId" ' ' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer ' "$token" ' ' \
```

## » Exemplo de código em phyton

```
import requests, json
import base64

KEY="XXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
SECRET="XXXXXXXXXXXXX"

SENHA = base64.b64encode( (KEY + ':' + SECRET ).encode('ascii') ).decode('ascii')

GRANT_TYPE = "client_credentials";

URL = "https://api.claro.com.br";
END_POINT_ACCESS_TOKEN = "/oauth2/v1/token";
END_POINT_API_FACIALBIOMETRICS = "/identity/v1/facialbiometrics";

headers={'Content-Type':'application/x-www-form-urlencoded','cache-cont
rol':'no-cache','x-client-auth': 'Basic {}'.format(SENHA)}

data = {'grant_type': 'client_credentials'}
respToken = requests.post(URL + END_POINT_ACCESS_TOKEN ,
headers=headers , data= data)

if respToken.status_code != 200:
    print('Erro: ' + str( respToken.status_code))
else:
    token = json.loads( respToken.text)['access_token']

    identificationId = "98676489009"
    identificationType = "CPF"
```

```
    headers_app = {'Content-Type': 'application/json',
                   'X-QueryString': 'identificationType=' + identificationType + '&IdentificationId=' +
                   IdentificationId,
                   'Accept':'application/json',
                   'cache-control':'no-cache',
                   'user-agent':'curl/7.60.0',
                   'Authorization': 'Bearer {}'.format(token)}

   resp = requests.get(URL + END_POINT_API_FACIALBIOMETRICS,
headers=headers_app)
   print (vars (resp))

   if resp.status_code != 200:
       print('Erro: ' + str(resp.status_code))
    else:
       print('Válida ' + str( resp.text ))
```

## » Exemplo de código em php

```
-

<?php

$key = 'XXXXXXXXXXXXXXXXXXXX';

$secret = 'XXXXXXXXXXXXX';

$identificationType = 'CPF';
$identificationId = '98676489009';

$token = getAccessToken();

$score = getScore($token , $identificationType, $identificationId);


function getAccessToken() {

    try{

            $senha = base64_encode($GLOBALS['key'] . ':' .
$GLOBALS['secret']);

            $ch = curl_init();
            curl_setopt($ch, CURLOPT_URL,
"https://api.claro.com.br/oauth2/v1/token");
            curl_setopt( $ch, CURLOPT_HTTPHEADER, [
                'Content-Type: application/x-www-form-urlencoded',
                'X-CustomerID: claro_exemplo',
                'x-client-auth: Basic ' . $senha

            ]);
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
            curl_setopt($ch, CURLINFO_HEADER_OUT, true);
            curl_setopt($ch, CURLOPT_POST, true);
            curl_setopt($ch, CURLOPT_POSTFIELDS,
'grant_type=client_credentials');

            $response = curl_exec($ch);
```

```
            if (!$response){
                $error = curl_error($ch);
                $info = curl_getinfo($ch);
                die("cURL request failed, error = {$error}; info = " .
print_r($info, true));
            }
            if(curl_errno($ch)){
                curl_close($ch);
                echo 'error:' . curl_error($ch);
            } else {
                curl_close($ch);
                return json_decode($response)->access_token;
            }

    } catch (Exception $e) {
        return 'Erro '. $e;
    }
}


function getScore($token , $identificationType, $identificationId){

    try {

        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL,
"https://api.claro.com.br/identity/v1/facialbiometrics");
        curl_setopt( $ch, CURLOPT_HTTPHEADER, [
            'Content-Type: application/json',
            'X-QueryString: identificationType=' .$identificationType. '&identificationId=' .$identificationId,
            'Accept: application/json',
            'Authorization: Bearer ' . $token

        ]);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLINFO_HEADER_OUT, true);

        $response = curl_exec($ch);

        if (!$response){
            $error = curl_error($ch);
            $info = curl_getinfo($ch);
            die("cURL request failed, error = {$error}; info = " .
print_r($info, true));
        }
        if(curl_errno($ch)){
            curl_close($ch);
            echo 'error:' . curl_error($ch);
        } else {
            curl_close($ch);
            return $response;
        }

    } catch (Exception $e) {
        return 'Erro '. $e;
    }
}
```

## » Exemplo de código em java

```
package client.facialBiometrics;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.Base64;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestClientException;
import org.springframework.web.client.RestTemplate;


public class App {

        private static final String KEY = "XXXXXXXXXXXXXXXXXXXXXXXXXX";
        private static final String SECRET = "XXXXXXXXXXXX";
        private static final String GRANT_TYPE = "client_credentials";
        private static final String URL = "https://api.claro.com.br";
        private static final String END_POINT_ACCESS_TOKEN =
"/oauth2/v1/token";
        private static final String END_POINT_API_FACIALBIOMETRICS =
"/identity/v1/facialbiometrics";


   public String getFacialMatch (String accessToken, String identificationId) throws RestClientException,
URISyntaxException {

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        headers.add("cache-control","no-cache");
        headers.add("x-client-auth", "Bearer " + acessToken);
        headers.add("X-QueryString", "identificationType=CPF&identificationId=" + identificationId);

        HttpEntity<String> request = new HttpEntity<>(headers);
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<String> result = restTemplate.getForEntity(new URI(URL +
END_POINT_API_FACIALBIOMETRICS), request, String.class);

        return result.getBody();
   }

   public String getAccessToken() throws URISyntaxException, JsonProcessingException, IOException {
        String senha = Base64.getEncoder().encodeToString( (App.KEY + ":" + App.SECRET) .getBytes() );
        HttpHeaders headers = new HttpHeaders();

        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        headers.add("cache-control","no-cache");
        headers.add("x-client-auth", "Basic " + senha);
        MultiValueMap<String, String> parameters = new LinkedMultiValueMap<>();
        parameters.add("grant_type", GRANT_TYPE);
        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(parameters, headers);

        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<String> result = restTemplate.postForEntity(new URI(URL +
END_POINT_ACCESS_TOKEN),  request, String.class);

        String responseBody = result.getBody();
        if (responseBody != null && !responseBody.isEmpty () ) {
ObjectMapper().readTree(responseBody).get("access_token").asText() ;
   } else {
        throw new IllegalStateException ("Response body is null or empty")
        }
   }
}
```