

MANUAL DE DESENVOLVEDOR

BRADESCO

Conexão às APIs – Interface de Programação de Aplicativo

Sumário

OBJETIVO	1
RESUMO	1
1. GERAR ID DA APLICAÇÃO	2
2. GERAR PAR DE CHAVES	2
3. OBTER ACCESS-TOKEN	5
3.1 Gerar JWT	5
3.2 Request para obtenção do Access-Token	8
4. CHAMADA A API DO SERVIÇO	11
4.1 Gerar Arquivo request.txt	11
4.2 Gerar assinatura request.txt	15
4.3 Realizar chamada à API do serviço	17
5. SUPORTE	18
6. PRODUÇÃO	18

OBJETIVO

Este manual apresentará o modelo de acesso às APIs – Interface de Programação de Aplicativos da Organização Bradesco, com foco no processo de autenticação e autorização de aplicações servidor ao servidor. Será demonstrado o passo a passo para automatizar o uso das APIs.

Nesse modelo, a autorização de acesso considerará os recursos acessados pertencentes à aplicação servidora e o token de acesso será emitido para a própria aplicação, e não para um usuário final.

O padrão de autorização adotado será o *JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*.

RESUMO

Para atender o objetivo acima, o manual apresentará como:

1. Gerar/Obter o ID da aplicação;
2. Gerar o par de chaves – privada e pública;
3. Gerar o JWT assertion para requisição do Access-token;
4. Obter o Access Token JWT;
5. Gerar a assinatura do request de serviço;
6. Chamar o serviço de exemplo – (/v1.1/jwt-service).

PASSO A PASSO

1. GERAR ID DA APLICAÇÃO

Para geração do ID da aplicação é necessário enviar os dados da Empresa (Nome, CNPJ, dados para contato) e informações sobre seu Aplicativo/Website (finalidade e endpoints serão utilizados) juntamente com a chave pública gerada conforme detalhado nos itens abaixo para o seguinte e-mail:

plataforma.api@bradesco.com.br

2. GERAR PAR DE CHAVES

Para gerar a chave privada e a chave pública autoassinada, use o OpenSSL. Recomenda-se usar a versão mais atual.

Observação:

O OpenSSL pode ser executado em um ambiente Linux(ex: MobaXterm, Putty) ou através do git-bash.exe instalado com o Git client.

O par de chaves deverá ser gerado com a validade **1125 dias** (aproximadamente 3 anos) com o tamanho mínimo de **2048 bits**, usando um dos algoritmos suportados:

- **RS256** – RSASSA-PKCS1-v1_5 usando SHA-256;
- **RS384** – RSASSA-PKCS1-v1_5 usando SHA-384;
- **RS512** – RSASSA-PKCS1-v1_5 usando SHA-512.

Para gerar, como exemplo, execute o seguinte comando:

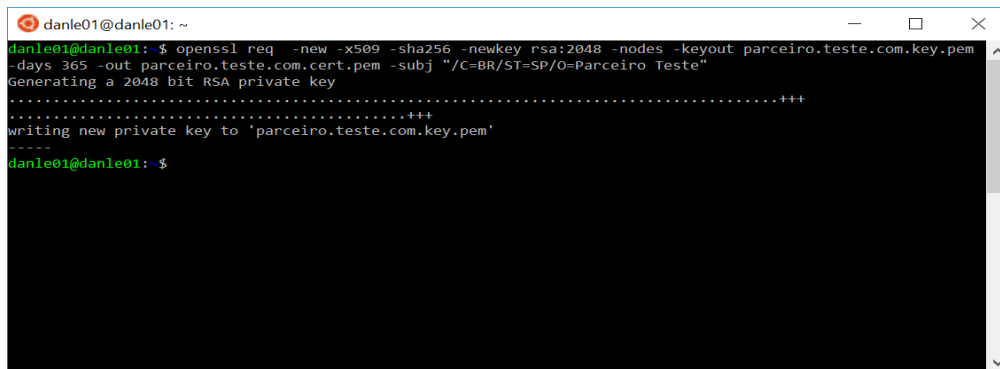
```
openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout <nome do arquivo da chave privada> -days <vigência da chave> -out <nome do arquivo do certificado> -subj <subject DN do certificado>
```

Como boas práticas recomenda-se utilizar no lugar de “parceiro” o nome associado da sua organização.

Exemplo:

```
openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout parceiro.teste.com.key.pem -days 365 -out parceiro.teste.com.cert.pem -subj "/C=BR/ST=PR/O=Parceiro teste"
```

Observações:



```
danle01@danle01: ~  
danle01@danle01:~$ openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout parceiro.teste.com.key.pem  
-days 365 -out parceiro.teste.com.cert.pem -subj "/C=BR/ST=SP/O=Parceiro Teste"  
Generating a 2048 bit RSA private key  
.....+++  
writing new private key to 'parceiro.teste.com.key.pem'  
-----  
danle01@danle01:~$
```

- Caso se depare com o seguinte erro:

Erro:

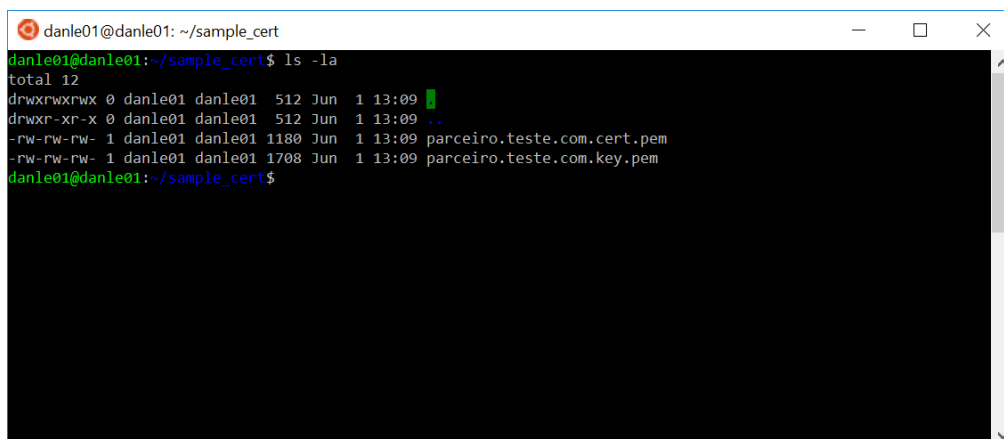
name is expected to be in the format /type0=value0/type1=value1/type2=... where characters may be escaped by \.

Utilizar a variável “MSYS_NO_PATHCONV=1” antes da execução, ficando conforme abaixo.

Exemplo:

MSYS_NO_PATHCONV=1 openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout parceiro.teste.com.key.pem -days 365 -out parceiro.teste.com.cert.pem -subj "/C=BR/ST=PR/O=Parceiro teste"

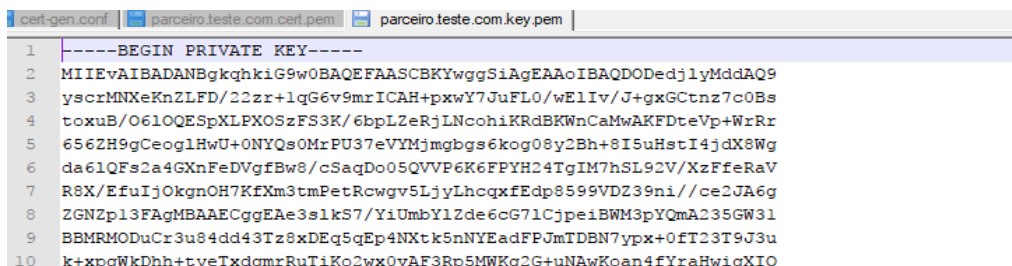
- Cada execução do comando, embora tenha os mesmos parâmetros, gera uma nova chave privada em um novo certificado. A chave pública fica incorporada no certificado gerado.
- Esse comando irá gerar dois arquivos: a chave privada e a chave pública, em formato PEM (Base64).



```
danle01@danle01: ~/sample_cert  
danle01@danle01:~/sample_cert$ ls -la  
total 12  
drwxrwxrwx 0 danle01 danle01 512 Jun 1 13:09 .  
drwxr-xr-x 0 danle01 danle01 512 Jun 1 13:09 ..  
-rw-rw-rw- 1 danle01 danle01 1180 Jun 1 13:09 parceiro.teste.com.cert.pem  
-rw-rw-rw- 1 danle01 danle01 1708 Jun 1 13:09 parceiro.teste.com.key.pem  
danle01@danle01:~/sample_cert$
```

Para conferir a chave privada, abra o arquivo em um editor de texto. A primeira linha do arquivo deve ter o texto “BEGIN PRIVATE KEY”.

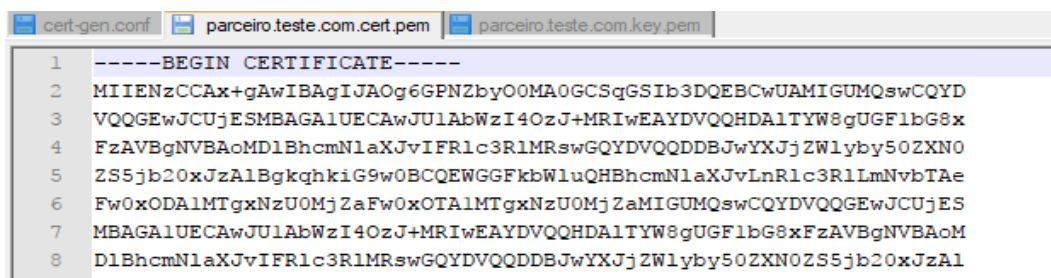
Exemplo: parceiro.teste.com.key.pem



```
1 -----BEGIN PRIVATE KEY-----
2 MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBywggSiAgEAAoIBAQDODedjlyMddAQ9
3 yscrMNXeKnZLFD/22zr+1qG6v9mrICAH+pxwY7JuFL0/wELIv/J+gxGctnz7c0Bs
4 toxuB/O6lOQESpXLPXOSzFS3K/6bpLZeRjLNcohiKRdBKWnCaMwAKFDteVp+WrrR
5 656ZH9gCeog1HwU+0NYQs0MrPU37eVYMjmgbs6kog08y2Bh+8I5uHstI4jdX8Wg
6 da6lQFs2a4GXnFeDVgfBw8/cSaqDo05QVVP6K6FPYH24TgIM7hSL92V/XzFfeRaV
7 R8X/EfuIjOkgnOH7KfXm3tmPetrCwgv5LjyLhcqxfEdp8599VDZ39ni//ce2JA6g
8 ZGNZp13FAGMBAAECggEAe3s1kS7/YiUmbY1Zde6cG71CjpeiBWM3pYQmA235GW3l
9 BMMRMODuCr3u84dd43Tz8xDEq5qEp4NXtk5nNYEadFPJmTDBN7ypx+0ft23T9J3u
10 k+xpGwkdDhh+tjeTxdgmRruTiKo2wx0vAF3Rp5MWKq2G+uNAwKoan4fYraHwigXIO
```

Na chave pública, o texto deve começar com “BEGIN CERTIFICATE”.

Exemplo: parceiro.teste.com.cert.pem



```
1 -----BEGIN CERTIFICATE-----
2 MIIEHzCCAax+gAwIBAgIJAQg6GPNZbyO0MA0GCSqGSIb3DQEBCwUAMIGUMQswCQYD
3 VQOGEwJCUjESMBAGAwJULAbWzI4OzJ+MRIwEAYDVQQHDA1TYW8gUGFlbG8x
4 FzAVBgNVBAoMDlBhcnNlaXJvIFRlc3R1MRswGQYDVQDDBJwYXJjZW1yby50ZXNO
5 ZS5jb20xJzAlBgkqhkiG9w0BCQEWGGfkbWluQHhcnNlaXJvLnRlc3R1LmNvbTAE
6 Fw0xODAlMTgxNzU0MjZaFw0xOTA1MTgxNzU0MjZaMIGUMQswCQYDVQOGEwJCUjES
7 MBAGAwJULAbWzI4OzJ+MRIwEAYDVQQHDA1TYW8gUGFlbG8xZzAVBgNVBAoM
8 DlBhcnNlaXJvIFRlc3R1MRswGQYDVQDDBJwYXJjZW1yby50ZXNOZS5jb20xJzAl
```

OBSERVAÇÃO:

Este arquivo é um certificado que tem a chave pública e alguns metadados adicionais, como o nome de quem gerou o par de chaves, entre outros.

A chave privada é de responsabilidade de sua empresa e deve ser armazenada de forma segura e nunca deverá ser fornecida a terceiros. A chave pública, e **somente a chave pública**, deve ser enviada em dois e-mails separados da seguinte maneira:

1. Chave pública zipada com senha;
2. Senha de descompactação em um arquivo “.txt” em anexo.

Os e-mails devem ser destinados para a caixa: plataforma.api@bradesco.com.br

Após a resposta com o fornecimento do ID de acesso, a utilização no ambiente destinado já pode ser iniciada.

OBS: Para o consumo das APIs, utiliza-se duas requisições.

- Obtenção do access-token;
- Consulta ao endpoint do serviço;

Os passos seguintes irão descrever como realizar estas duas requisições de forma “manual” utilizando-se a ferramenta Postman, os fluxos devem ser desenvolvidos para funcionar de forma automatizada nos sistemas consumidores.

3. OBTER ACCESS-TOKEN

3.1 Gerar JWT

O JWT é utilizado na primeira requisição para obtenção de token de acesso.

O desenvolvedor deverá gerar um JWT de acesso e fazer a assinatura com um dos algoritmos indicados (ex: RS256) com sua chave privada. O JWT é formado por dois JSONs, sendo eles "header.json" e "payload.json".

- **Header.json:**

```
{
  "alg": "<algoritmo utilizado>",
  "typ": "JWT"
}
```

Exemplo de preenchimento:

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

- **Payload.json:**

```
{
  "aud" : "<endereço do serviço em que o token será gerado>",
  "sub" : "<id client do cliente >",
  "iat" : "<data de geração deste jwt, no formato NumericDate>",
  "exp" : "<data de expiração deste jwt, no formato NumericDate>",
  "jti" : "<nonce – numérico(18) random, a cada chamada deve ser trocado, ex: unix current date in milliseconds >",
  "ver" : "1.1"
}
```

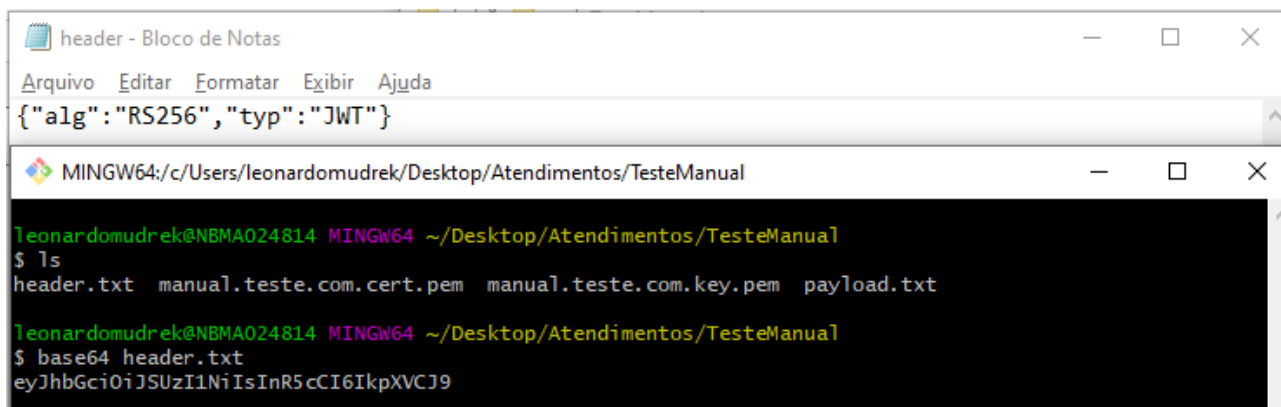
Exemplo de payload preenchido:

```
{
  "aud" : "https://proxy.api.prebanco.com.br/auth/server/v1.1/token",
  "sub" : "bc7ccf09-8a85-4be6-y67e-82bf11737994", <id cliente fornecido pelo banco>
  "iat" : "1574094116",    <data atual em segundos>
  "exp" : "1576686116",    <data atual adicionando 1 mês à frente, em segundos>
  "jti" : "1574094116000", < numérico(18) random, a cada chamada deve ser trocado, ex: unix current date in milliseconds >
  "ver" : "1.1"
}
```

Com estes dois JSONs formatados, ou seja, retirado todos os espaços e quebras de linhas (conhecido como JSON.stringify), realize o encode de ambos para base64.

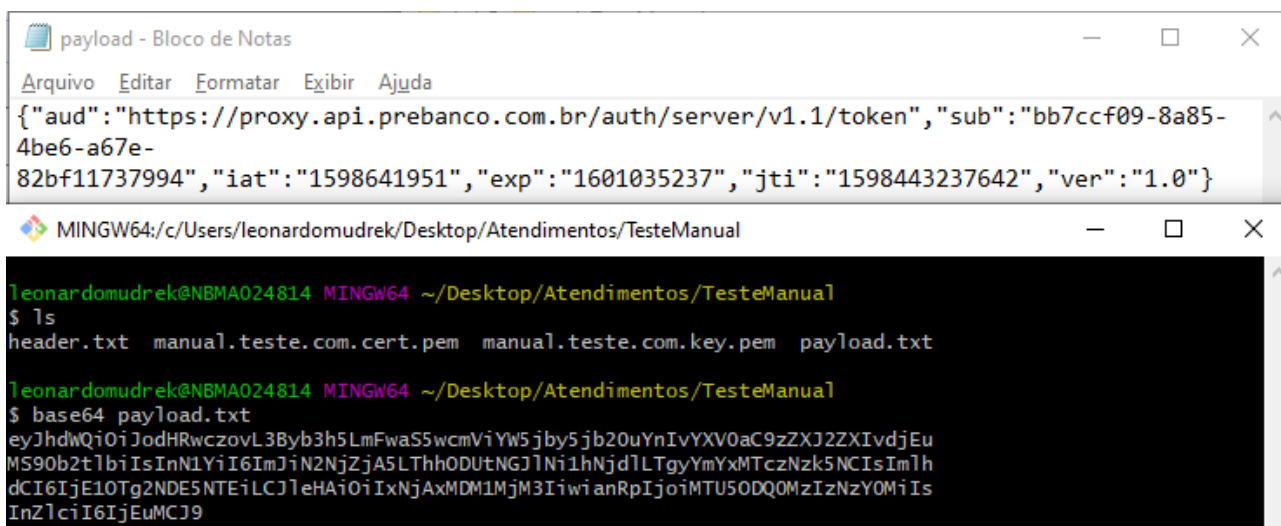
```
base64 <nome_do_arquivo>
```

Header:



The screenshot shows two windows. The top window is a Notepad titled 'header - Bloco de Notas' containing the JSON: `{"alg": "RS256", "typ": "JWT"}`. The bottom window is a terminal titled 'MINGW64: c:/Users/leonardomudrek/Desktop/Atendimentos/TesteManual'. It shows the command `ls` listing files: `header.txt manual.teste.com.cert.pem manual.teste.com.key.pem payload.txt`. Then, it shows the command `base64 header.txt` being executed, resulting in the output: `eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9`.

Payload:



The screenshot shows two windows. The top window is a Notepad titled 'payload - Bloco de Notas' containing a large JSON object: `{"aud": "https://proxy.api.prebanco.com.br/auth/server/v1.1/token", "sub": "bb7ccf09-8a85-4be6-a67e-82bf11737994", "iat": "1598641951", "exp": "1601035237", "jti": "1598443237642", "ver": "1.0"}`. The bottom window is a terminal titled 'MINGW64: c:/Users/leonardomudrek/Desktop/Atendimentos/TesteManual'. It shows the command `ls` listing files: `header.txt manual.teste.com.cert.pem manual.teste.com.key.pem payload.txt`. Then, it shows the command `base64 payload.txt` being executed, resulting in a long base64 encoded string: `eyJhdwQiOiJodHRwczovL3Byb3h5LmFwaS5wcmViYW5jb20uYnIvYXV0aC9zZXJ2ZXIvdjEuMS90b2t1biIsInN1YiI6ImJiN2NjZjA5LTlhODU0TNGJlNi1hNjd1LTgyYmYxMTczNzk5NCIsIm1hZCI6IjE1OTg2NDE5NTEiLCJleHAiOiIxNjAxMDM1MjM3IiwianRpIjoiaMTU5ODQ0MzIzNzY0MiIsInZlciI6IjEuMCJ9`.

Concatene os dois resultados separando-os com o caractere “.” (ponto); O formato será:

Header(base64) + “.” + Payload(base64)

Para gerar a assinatura utilize o seguinte comando:

```
echo -n "$(cat <Arquivo_de_assinatura.txt>)" | openssl dgst -sha256 -keyform pem -sign  
<nome_da_chave_privada.key.pem> -out <arquivo_de_saida.txt.256>
```

Exemplo de execução utilizando o arquivo “headerAndPayload.txt” e chave privada “manual.teste.com.key.pem”:

3.2 Request para obtenção do Access-Token

O *access-token* é de uso único e deve ser gerado a cada nova chamada das APIs. Para realização do teste, recomendamos usar o **Postman**.

Antes de criar a requisição, certifique-se que o “**SSL certificate verification**” no Postman está **DESATIVADO!**

File → Settings → General → SSL certificate verification

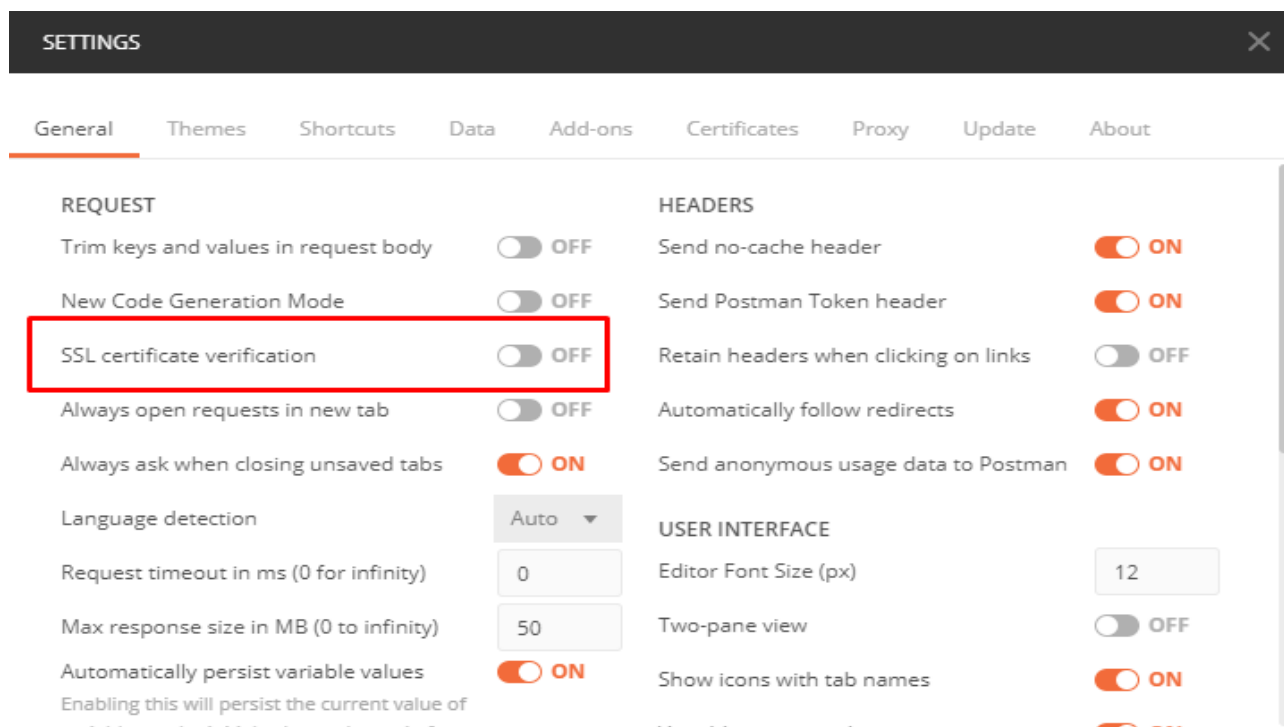


Figura 1. DESATIVANDO SSL CERTIFICATE VERIFICATION

A requisição do Postman para obtenção do token deverá conter as informações de “método” e “headers” conforme abaixo para a URL de serviço:

<https://proxy.api.prebanco.com.br/auth/server/v1.1/token>

Method POST

URL: <https://proxy.api.prebanco.com.br/auth/server/v1.1/token>

Body: selecionar “x-www-form-urlencoded”

Key: grant_type = urn:ietf:params:oauth:grant-type:jwt-bearer

Key: assertion = <JWTgerado>

Exemplo da request no Postman:

POST ▼ https://proxy.api.prebanco.com.br/auth/server/v1.1/token

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings

☐ none
 ☐ form-data
 ☒ x-www-form-urlencoded
 ☐ raw
 ☐ binary
 ☐ GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	grant_type	urn:ietf:params:oauth:grant-type	
<input checked="" type="checkbox"/>	assertion	eyJhbGciOiJIUzI1NiIsInR5cCI6Ikpz	Inserir JWT gerado

Figura 2. EXEMPLO DE REQUEST POSTMAN

O Postman adicionará o header Content-Type automaticamente:

POST Gerador de Acesso_Token + ... No Environment ▼ 👁 ⚙

► Gerador de Acesso_Token Comments (0) Examples (0) ▼

POST ▼ https://gateway:8443/auth/server/v1/token Send ▼ Save ▼

Params Authorization **Headers (1)** **Body** ● Pre-request Script Tests Settings Cookies Code

▼ Headers (1)

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/>	Content-Type	application/x-www-form-urlencoded				
	Key	Value	Description			

Figura 3. HEADER CONFIGURADO PELO POSTMAN

```
{
  "access_token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.ew0KICJpc3MiOiAiaHR0cHM6Ly8yMDEuNy4xMjcuaMTcxOjg0NDMvYXV0aC9zZXJ2ZXLvdjEvdG9rZW4iLA0KICJhdWQlOiAiaHR0cHM6Ly9sYWI5LnByZWJhbmNvLmNvbS5icjo4NDQzliwNCiAiZXhwIjogMTUzNzI4MTIwOSwNCiAiaWF0IjogMTUzNDY4OTM1NSwNCiAic2NwljogIm9vYiIsDQogInZlcil6lClxLjAiLA0KICJhdXR0VHlwZSI6ICJzZXJ2ZXIIiLA0KICJhdXR0RGF0YSI6ICJleUowZVhBaU9pSktWMVFpTENkamRla2lPaUplVjFRaUxDsmhiR2NpT2IKU1UwRXRUMEZGVUNJc0ltVnVZeUk2SWtFeU5UWKRra010U0ZNMU1USWlmUS5BSjlXWWJkSDE2UDRPYIBJaGFmV0hqMldBX0tFQTVSZnRIVzlQQWFQVHFYm2tjamktMHJ5NFNSX2Jwb3p1RnJGR0MzNEY2bvdtdeIE1VFJ3dG5pSEpVVGnhQk5jeUIVYzd2cEctZWZyWWsxVzNKAERzMtF4TUZ3OFN1Sk9CRk14M0tDZGM4STBKMgdON1FTVnlCSIIKCEfITE83VTJsMHphQzc3YzIZY3dSLUFsZ1ZleUZzaHE4cHIqU0VaZXJtMDJLS1MwMTBJN09iVE2WWgtNHRDQktSQ1VPT2Z4bnhnZFp2RXg2UUxNOHByeE81OTJSNmFCY0Fkb1ImM3JHRmdielQ0YkF0RjdkbXBmZ2pacnZft2stNku1dU9DVjVyZ1JSNEFMkZ1Y0ZENHHKNWs2Mlp5Qm5FOS1sMU96QkJKWVMtS09fS3pfa0RQWGFAQ2hya3ctb3ZWS3IBOXdhS0dPN1hg3l3eUl6VXk5dTZZSlpHNfVwWGxxZ3R5MWdKZFRftZlKc0NISU1zZldHYVU1S3RnendaWkxER2N0Q0N3eFZsejJMM1Vxd1loTlgxWjhCR3Q1NVIOaUw1djRFNZVBURWZ6SzZ2a2U4WE5SZmdGcW5sSIJ4eW9nWE5EZ1JQNNG41dGJJOVBLTMJhWII0SkVscXpGbWhBEZelcjVqOHNNZ1Z2cnNHVHhJeIR0Y0ZMdUJCmfJMkg3SFNQSXRIOW96eGRRaGZtN01lcGctQ0VldjhRTZdpalTjd19jQTF5a043cUdPaUxaNGR2OU4xT3NWrd2piRmpXRmIEZUXVTXJHX0tvc0VIRkFYc3JMZzFOUjVGLURSUDIzeERLRkZJTzY1ektmTmZvTGt2Vld4bnFyckdVhGx5aXI2dVhhZlZTHNSM3ViWXN1MjZNWS55cGkyWmdybFl6F6QUR1UEVyoGtVNWDblNr2am5KMkvQOXVPMXRZcGtZbnIPY0JHTUJkdGh1Znd0NHJuT3h6b1BacUotbmRtSXELVFIInpuYjVtUE1Va3pCOTN3MXdwU2hldTVXdlFqdUFwNUtyd2RyaEtmRUdmRjFmUjVaYXIHVINrYnZLeXp2X2Jan2h5U21RZTVPT0FyaEhxbmF1MzRONFNouU0RuTmdKX1ZFBUdnVmNFMEEJPQ1VpY1hHMjROMDZZLUw3UnZQR18xakY5S01xWE14OWd3YmREeEpFOEFzUzIdDYy1aSfd2SHozemIXV2hKdDixTVNPQWxZeVfoNWR0TXpDYmw1ZFRzeHNaeTEwQzhwaEs2UG1QemZZQndLS1ItS05WV1U1VXNyZEpqVHPaHBISUNDCTd5UWHvSkFYy2dXNm1tYTdadGjkREllSEt3ay0tZTVfbnpht1NReTB0N3owUmQtSEdOWDZGYnNhRml0dFJUUV1paQmFXNTI5X0hPcENfRUkwNE5ta0daOC15MkdubzlbSThtdFRUUTZTVdg2R0JFYVdjWENIODhBenHOOTI6M052dTdLSG9wYndzR1luTEZnYXoxdTROaWtFejdCdFhQVGR1bdZRZ3VJaDRKmZJWNGIWZjIEVEhKYmJDalFweAXaVExLUkZJcm51VDRrZFZBWVdpQS1KNhk4enRsUU93V290TFNUQkl2TkplJanYxeS1ZUTc3QIFkYTIJNBtdWlucUx4ay1ja0l5QnNXRG1lQW9xcckdlUy1tT0puR1VNYzVMbkYzN1k4RU9fVUh2WGc4MVlhdfZWZ2ZFWHVHUU2djNGFPV1JtYVFXv1lzV1V5YUttbTNvcjBqa0pXaE9sbkFQV1JnSk13dVpsQIFPa2hfVmVScTFacFQyUFJWW14emVpcEh0MnNodzZGd3l3MlotZ2NrTmk2a3NEOWHTNDJWaHd1c3RtNlVUvo2OUtir2xqlTZHZI94d1B0SkILaGVTNDZoQ0RSOfAyMjVqSHIPZjF2QXZsZFpPWXFJdE5sWDZft1F0M20xMXRhSHpFdZluRm5ZaUxpcThSnMN0c0RWck5xYU8xSDZlX1J0Ymc1eEY2NmpQZk5PS2IfNDBXx1F4U2REQVhia003SDV2cFpLZlhjak5sNnFWYXNPbe1wnc1pWUVESFBqdGRsSGJpSIR3ZUtISjl3c0V1WIQ2VXFYtjdCOGI3cmxPTmNCZU4wdFBxbzJycTR3amJDY1I2ZElydTKZK1hvWiBpRINGZEFDaGRPUI4WDNfrVduWnp rSWffVd0SGY2X2ZlaUJMChMWOEXfwKRJaWQOLVRqRC1nMknISHdSR2pXTC1pVDdpdEV0U2Zus0g2d3c3SU1WV2d1VEI3UIhlbnPVSENUWjc0WVNkY3dEOdFEc0NLsktHQ0l4eVZIUIhRaGdPRHdyWjkwMUs2bHNxa2ItWTR3MktaY2hNaUFxUVZQSJRQQLdkbkd2RKvXSU9tbXdFUmlFRWczMKkzt0tyT183dEFjY1NrUWZxSUN5bU41LWNatTR1dIRBRjdHMKhfBjMXTetEOHCzyXFseWNBV2RWNExpT1ZYZXVgbmtJU1RrMFRaVDQuaTZ5Q0VoMIZSc1FIWUFzVTBkOUPlTWdDa2FyUV9HWm9Fa3ZrNmkt1ViNCINCn0.Rr6C_uCohQMlmT0fCv-HUMn-WLoRfgwP4euYbJz2plsm_CsSwWegZn3XQWbWGC7QuAL6wdkMOkoSeOQBwdNUNukyjuDVXJEh_W6e-wvBaE8TWrsS-
    CijJ9X9Xdp93uObT99YaYSmGj22bLGtGdl86su7GYI5ORzoFKyYkuACsi1Q0AbqLbrKugYgnFrSFGzatkcjO9eeKqsJ4xUdPIEbGBbRLdEntkcs3MPNU-XKT9eGv1opOg1JkKciibOfcP1HJHDiAcFHxLqmPGhrVJ8Wu99y3nWwzfvyj9MilK7OjJ_63ROZQyTV-Fb7jbh0sGA9i3TrcQGGrnfJ_CCayfctC4ah4KrF02J-UcQ1xs52ezeNdvlJflgsKpZT8dfN-vpgyxuJTR9mq9HDCGJQyjf-ivoaf7CnTy8peSLY9KnzxlvVPdBAgeKh7qOn058-Tzu2Y06d8mEfd3xet_m1wE4A85-7anbw2uKI1PTAQIEmpuY46xAH1FSXq97fcUYaeE7Z695if_yipyZsWbntOnU-9dzCm4tu76FnDc45Q6qd7UqDCXDSSFnxhe9eqX5dCIIO3eDgU09_pCd5qAQypRLqTkWe5ib-8aG5jotBsPENQNnxie57ol8DR2y_k-dCjwPQ1RRjYHM9Xd8gqeivP6WnAx4wr2kfFo8FNndfLpo",
  "token_type": "Bearer",
  "expires_in": 2591854.0,
  "scope": "oob"
}
```

OBSERVAÇÃO: Este access-token será utilizado para realizar as chamadas nos endpoints dos serviços. A geração de um novo access-token só deve ser feita após o mesmo expirar, conforme data recebida no retorno no atributo “**expires_in**”. O efeito colateral de gerações continuas é se deparar com o erro abaixo.

```
{
  "code": 0,
  "message": "unexpected error",
  "details": [
    {
      "name": "internalCode",
      "value": "FRWK0103"
    },
    {
      "name": "internalMessage",
      "value": "EXCEDIDO O LIMITE DE SESSÕES ABERTAS SIMULTANEAS PARA ESTE
USUÁRIO. PARA ABRIR UMA NOVA SESSÃO, FECHER A SESSÃO QUE ESTÁ ATIVA NO
TERMINAL APIFRONT OU UMA SESSÃO ATIVA EM OUTRO TERMINAL."
    }
  ]
}
```

4. CHAMADA A API DO SERVIÇO

Neste manual utilizaremos o endpoint de exemplo (/v1.1/jwt-service). A resposta do recurso é “API acessada com sucesso!” e pode ser utilizada para o teste de autenticação na camada de segurança.

4.1 Gerar Arquivo request.txt

Um dos passos é gerar uma string a ser assinada pela chave privada. Neste manual será criado um arquivo com o nome request.txt. Porém em uma aplicação é necessário somente construir a string no formato correto, não sendo necessário de fato a criação de um arquivo.

Nesta string deverão ser incluídas algumas informações que serão utilizadas na chamada do postman tais como, método, URL, parâmetros, endpoint e inclusive o access token obtido anteriormente no serviço:

<https://<endereço do ambiente>/auth/server/v1.1/token>

As informações no arquivo devem ser inseridas assumindo que cada informação nova assumirá uma nova linha. O arquivo request.txt deverá estar no mesmo diretório que a chave pública e a chave privada e seguir o seguinte padrão:

VERBO^[1] - linha 1
<URI da chamada>^[2] - linha 2
<Parâmetros que estão sendo utilizados na URL>^[3] - linha 3
<body>^[4] - linha 4...
<Valor do access-token>^[5]
<Nonce>^[6]
<Timestamp>^[7]
<Algoritmo que está sendo utilizado>^[8]

É possível que os “parâmetros” e token ocupem várias linhas do arquivo, assim uma nova informação deverá ser colocada logo abaixo da outra. A ordem de informações no arquivo deve permanecer exatamente conforme o modelo acima.

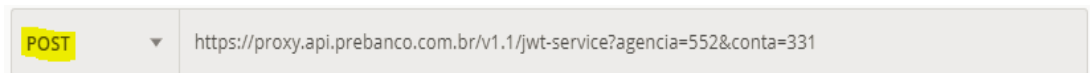
O arquivo “request.txt” depois de preenchido e devidamente assinado, será utilizado na chamada ao endpoint de desejo.

Segue exemplo de preenchimento do arquivo que será assinado e de como ficará a chamada no Postman.

- Linha 1: método utilizado no Postman.

POST^[1] - linha 1 do arquivo

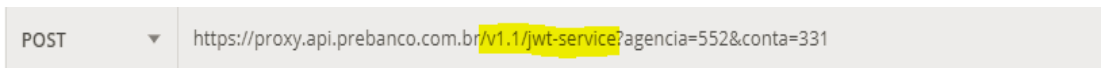
Colocaremos este mesmo verbo na chamada Postman:



- Linha 2: URI da chamada (endpoint à API consultada)

/v1.1/jwt-service^[2] - linha 2 do arquivo

E no postman:



- Linha 3: exemplo de parâmetros da chamada

agencia=552&conta=331^[3] - linha 3 do arquivo

Então no Postman temos os parâmetros que estão sendo utilizados, podemos inserir esses valores na URI após o “?” ou na aba “params” do Postman.

Parâmetros:

POST ▼ https://proxy.api.prebanco.com.br/v1.1/jwt-service?agencia=552&conta=331

Aba params:

Params ● Authorization Headers (6) Body Pre-request Script Tests

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	agencia	552
<input checked="" type="checkbox"/>	conta	331
	Key	Value

- Linha 4: body da chamada

`{"teste": "valor"}[4]`

O body dessa chamada deve ser “raw” “JSON” no postman.

Params ● Authorization Headers (6) **Body ●** Pr

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw

1 `{ "teste": "valor" }`

No caso de um endpoint específico não possuir “parâmetros” ou “body” na requisição a linha do arquivo request.txt correspondente **deverá ficar em branco**.

- Linha 5: access-token gerado nos passos anteriores.

`eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.ew0KICJ...[5]`

No Postman devemos inserir esse valor no cabeçalho da chamada, que será referente a chave “Authorization”, e no campo valor deve iniciar com “Bearer[espaço] valor do access-token da seguinte forma:

Params ● Authorization **Headers (6)** Body ● Pre-request Script Tests

▼ Headers (6)

	KEY	VALUE
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.ew0KICJ...

- Linha 6: nonce (valor numérico(18) aleatório, que poderá ser utilizado uma única vez para cada chamada, neste caso está sendo utilizada a data atual em milissegundos).

`1574693951000[6]`

Este mesmo valor deve ser inserido no Postman, equivalente a chave “X-Brad-Nonce”, no “header” da chamada, ficando conforme imagem abaixo:

<input checked="" type="checkbox"/>	X-Brad-Nonce	1574693951000
-------------------------------------	--------------	---------------


- Linha 7: timestamp (refere-se a data e hora que está sendo efetuada a chamada para o endpoint). No caso do exemplo a data está sendo feita já no horário UTC - 3:00, caso contrário indicar variação em relação ao fuso brasileiro.

2019-11-25T11:23:00-00:00 ^[7]

Formato “AAAA-MM-DDThh:mm:ss-00:00”, sendo:

- AAAA = ano com quatro caracteres, exemplo “2019”, referindo-se ao ano atual;
- MM = mês com dois caracteres, exemplo “11”, referindo-se ao mês de novembro;
- DD = dia com dois caracteres, exemplo “25”, referindo-se ao dia 25;
- **T = texto fixo**;
- hh = hora com dois caracteres, exemplo “11”, referindo-se às 11 da manhã;
- mm = minutos com dois caracteres, exemplo “23”, referindo-se aos 23 minutos daquela hora;
- ss = segundos com dois caracteres, exemplo “00”;
- -00:00 = Diferença para o fuso horário UTC -3:00, no caso já está no fuso correto, então “-00:00”. Caso chamada utilizando fuso referencial UTC 0:00 (**2019-11-25T14:23:00-03:00**)

Esse mesmo valor deve ser inserido no cabeçalho da chamada, sendo a sua chave “X-Brad-Timestamp”, ficando no padrão ilustrado abaixo:

 <input checked="" type="checkbox"/>	X-Brad-Timestamp	2020-03-10T12:17:06-00:00
---	------------------	---------------------------

- Linha 8: algoritmo que está sendo utilizado.

SHA256^[8]

No cabeçalho da chamada o valor é correspondente a chave: “X-Brad-Algorithm”, conforme na imagem abaixo:

<input checked="" type="checkbox"/>	X-Brad-Algorithm	SHA256
-------------------------------------	------------------	--------

Então o arquivo para assinatura ficará da seguinte forma:


```
POST
/v1.1/jwt-service
agencia=552&conta=331
{"teste":"valor"}
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.ew0KICJ...
1574693951000
2019-11-25T11:23:00-00:00
SHA256
```

Observação: No exemplo acima estamos preenchendo tanto body quanto os parâmetros da requisição, porém, caso não seja utilizado body ou parâmetros deve-se deixar a linha referente ao que não está sendo utilizado em branco.

Abaixo ilustra como deve ser feito um arquivo de assinatura quando não está sendo passado o body na chamada:

```
POST
/v1.1/jwt-service
agencia=552&conta=331

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.ew0KICJ...
1574693951000
2019-11-25T11:23:00-00:00
SHA256
```

Note que nesse caso deixamos a linha referente ao body em branco.

OBSERVAÇÃO: É importante considerar a quebra de linha do padrão Unix (\n: LF – LineFeed) no arquivo request.

O padrão de quebra de linha do Windows (\r\n CR – CarriageReturn, LF- LineFeed) ou de algum outro sistema (\r CR – CarriageReturn) causarão uma assinatura inválida.

Ou seja, o arquivo deve ser salvo no padrão de quebra linha do Linux, caso for salvo em outro padrão de quebra linha, será retornado na chamada um erro de assinatura inválida.

4.2 Gerar assinatura request.txt

Então agora que o arquivo de assinatura está completo, será necessário assiná-lo, como exemplo utilizaremos o nome desse arquivo como “request.txt”.

Para assinar esse arquivo é necessário salvá-lo no mesmo diretório que estão as chaves pública e privada como mostra a imagem abaixo:

```
$ ls
request.txt  suporte.pem  suporte.teste.com.key.pem
```

Após isso execute o comando em um ambiente linux para gerar assinatura. A assinatura deve estar no padrão SHA256, o comando utilizado será o seguinte:

```
echo -n "$(catArquivo_de_assinatura.txt)" | openssl dgst -sha256 -keyform pem -sign  
nome_da_chave_privada.key.pem -out arquivo_de_saída.txt.256
```

Exemplo de execução do comando utilizando o arquivo “request.txt” e a chave privada “suporte.teste.com.key.pem”:

Exemplo:
`echo -n "$(cat request.txt)" | openssl dgst -sha256 -keyform pem -sign
suporte.teste.com.key.pem -out request.txt.256`

Após executado, não será retornado nenhuma mensagem pelo terminal, porém será criado um novo arquivo com extensão .256 que no nosso caso será o arquivo “request.txt.256”, conforme ilustra a imagem abaixo:

```
$ ls  
request.txt  request.txt.256  suporte.pem  suporte.teste.com.key.pem
```

Após isso será necessário codificar esse arquivo para base 64 utilizando o comando:

```
base64 <nome_do_arquivo>
```

Exemplo abaixo ilustra como codificar o arquivo “request.txt.256”:

Exemplo:
`base64 -wrap=0 request.txt.256`

Após executar esse comando será retornado a string da assinatura desse arquivo em base64, conforme ilustra imagem abaixo:

```
$ base64 --wrap=0 request.txt  
U1QKL2p3dC1zZXJ2aWN1CmFnZW5jaWE9NTUyJmNvb3R1PTZMOp7InRlc3R1IjoidmFsb3IifQpleUow  
ZVhBaU9pSktWMVFpTENKaGJHY2lPaUpTVXpVeElpSjkuZXcwS0lDSi4uLgoxNTc0NjkgOTUxMDAwCjIw  
MTkxMTI1VDExOjIzOjAwLTAwOjAwClNIQTIlIngoKIwo=  
[i392610@AT-CL-MT-023 teste-suporte]$
```

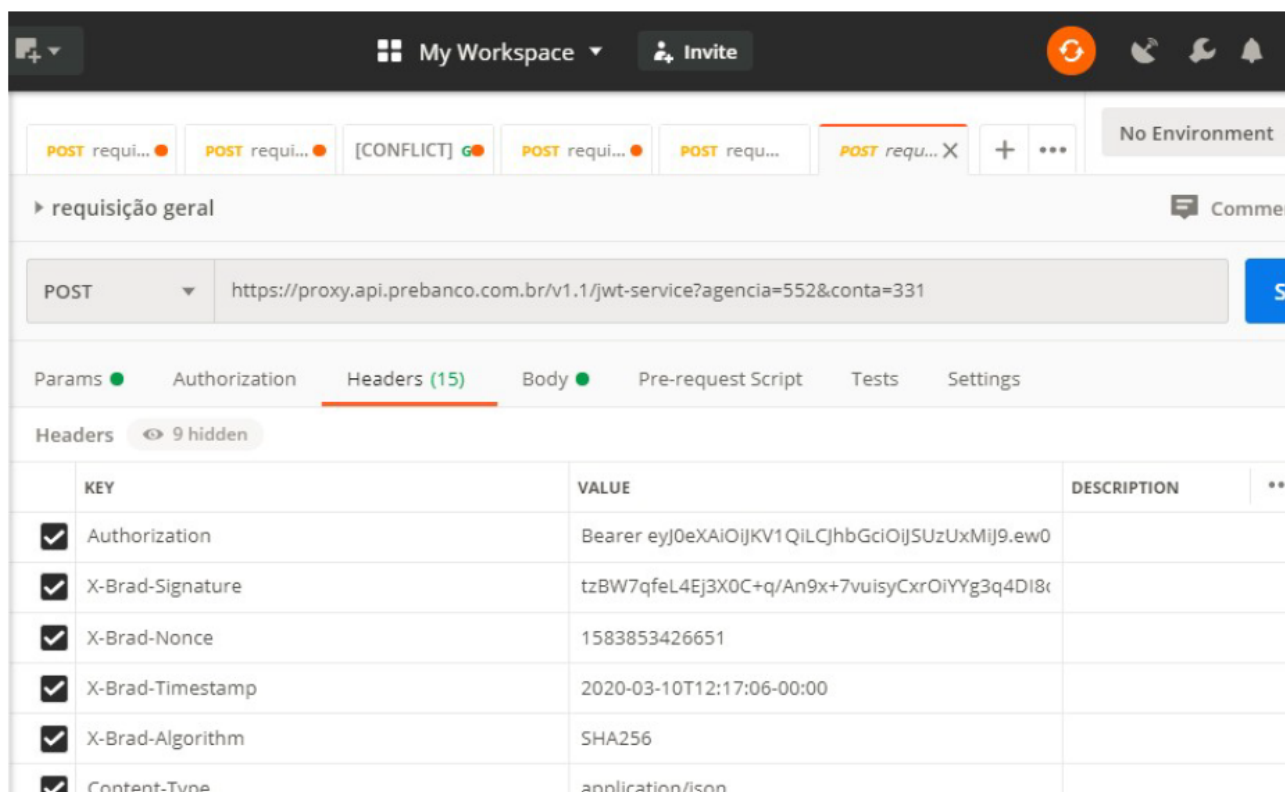
A string gerada será utilizada para fazer a chamada, essa String é equivalente à a chave “X-Brad-Signature” na chamada do postman.

<input checked="" type="checkbox"/> X-Brad-Signature	U1QKL2p3dC1zZXJ2aWN1CmFnZW5jaWE9NTUyJmNvb3R1PTZMOp7InRlc3R1IjoidmFsb3IifQpleUowZVhBaU9pSktWMVFpTENKaGJHY2lPaUpTVXpVeElpSjkuZXcwS0lDSi4uLgoxNTc0NjkgOTUxMDAwCjIwMTkxMTI1VDExOjIzOjAwLTAwOjAwClNIQTIlIngoKIwo=
--	--

Observação: Antes de colar a string no campo do valor do X-Brad-Signature é necessário deixá-lo em uma única linha, caso contrário sua chamada retornará erro de assinatura inválida.

4.3 Realizar chamada à API do serviço

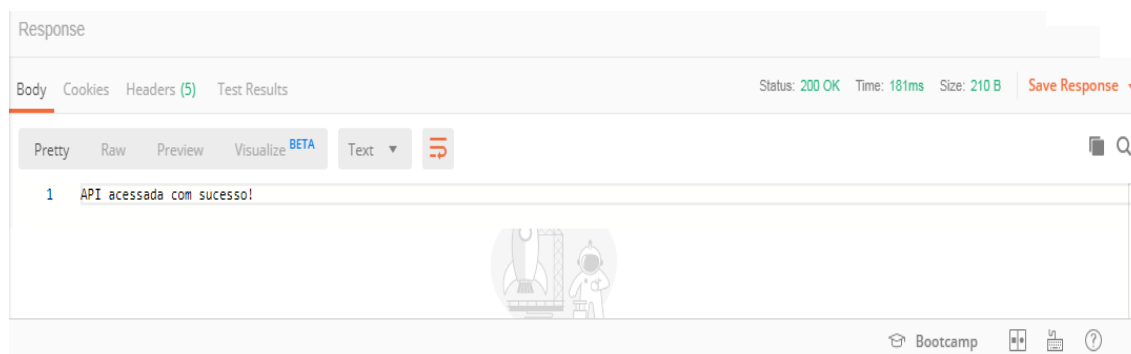
Então obtemos o cabeçalho completo do Postman da seguinte forma:



Explicação de cada campo:

Chave	Valor
Authorization	Bearer access-token
X-Brad-Signature	Valor da assinatura em base 64, em linha única
X-Brad-Nonce	Valor do nonce que foi inserido no arquivo de assinatura
X-Brad-timestamp	Timestamp inserido no arquivo de assinatura
X-Brad-Algorithm	Algoritmo utilizado

Após preenchidos os valores, a chamada a API “/v1.1/jwt-service” retornará a seguinte resposta: “API acessada com sucesso!”.



5.SUPORTE

Em caso de dúvidas ou necessidade de suporte, após seguir os procedimentos deste manual, entre em contato em nossa central de suporte pelo seguinte e-mail:

suporte.api@bradesco.com.br

Enviando as seguintes informações:

- CURL/Collection da requisição para geração do access-token;
- CURL/Collection da requisição para a API;
- String (request.txt) utilizado para assinar a chamada para a API;
- CNPJ e nome da empresa que contratou o serviço junto ao Bradesco

6.PRODUÇÃO

Após a conclusão dos testes em ambiente de homologação é necessário seguir alguns passos para que seja disponibilizado o acesso no ambiente.

- 1- Realizar a geração do novo par de chaves para o ambiente de produção. Data de expiração entre um e três anos.

Exemplo:

```
openssl req -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout  
parceiro.producao.com.key.pem -days 365 -out parceiro.producao.com.cert.pem -subj  
"/C=BR/ST=PR/O=Parceiro producao"
```

- 2- Realizar o envio da nova chave pública gerada. No formato de dois e-mails:

- i. Chave pública zipada com senha;
- ii. Senha de descompactação em um arquivo “.txt” em anexo

Os e-mails devem ser destinados para a caixa: plataforma.api@bradesco.com.br

- 3- Inserir no e-mail com a chave pública zipada, os seguintes dados:

- i. Evidência de teste realizado com sucesso na API consumida;
- ii. CNPJ e nome da empresa;
- iii. Ponto de referência para contactar durante o processo de renovação da chave pública;

OBS: O envio dos certificados para produção ocorre toda segunda-feira, estes serão cadastrados na sexta-feira da mesma semana. Sendo assim, envios feitos após a segunda-feira somente serão implantados na sexta-feira da semana seguinte.