

BANCO BRADESCO S.A
SUORTE API

MANUAL AOS DESENVOLVEDORES
Conexão às APIs – Interface de Programação de Aplicativo

2021

SUMÁRIO

1. OBJETIVO	3
2. GERAR ID DA APLICAÇÃO	4
2.1 CERTIFICADO PÚBLICO	4
2.1.1 AMBIENTE DE HOMOLOGAÇÃO.....	5
2.1.2 AMBIENTE DE PRODUÇÃO	6
2.2 ENVIO DADOS CADASTRAIS	7
3. CONSUMO APIs	8
3.1 OBTER ACCESS-TOKEN.....	8
3.1.1 JWT.....	9
3.1.2 JWS.....	11
3.1.3 REQUISIÇÃO HTTP	13
3.2 CONSUMO ENDPOINT	15
3.2.1 ASSINATURA.....	16
3.2.2 REQUISIÇÃO HTTP	22
4. SUPORTE.....	24
5. PRODUÇÃO.....	25

1. 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)*, relatado pela RFC 7523, "*Profile for OAuth 2.0 Client Authentication and Authorization Grants*".

2. GERAR ID DA APLICAÇÃO

Para a geração do ID da aplicação e assim utilização das *Open APIs*, se faz necessário o compartilhamento dos dados da empresa, juntamente com as informações sobre a aplicação a ser desenvolvida. O cadastro será através do certificado público digital fornecido, sendo um certificado para cada ambiente, homologação e produção. Características de construção e formato esperado são descritos a seguir.

2.1 CERTIFICADO PÚBLICO

O envio deve ser feito de toda a cadeia de certificados – raiz, intermediários e domínio/empresa, sendo extensão “.PEM” (base64) e padrão X.509 o formato requisitado. Na composição dos dados, se torna premissa a identificação da empresa, referendando CNJP e razão social. Os algoritmos suportados são:

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

O arquivo a ser disponibilizado, ao realizar a leitura em formato de texto, deve começar com “BEGIN CERTIFICATE”, conforme exemplo abaixo.

Quadro 01 – Exemplo arquivo certificado público, formato X.509.

Exemplo: manual.teste.com.cert.pem

-----BEGIN CERTIFICATE-----

MIIDQzCCAiugAwIBAgIUcByYmH6Btz/Fk3p5QMvedrO4+fgwDQYJKoZIhvcNAQELBQAwMT
ELMAkGA1UEBhMCQlxCzAJBgNVBAGMAIBSMRUwEwYDVQQKDAxUZXRNOZSBtYW51YW...

-----END CERTIFICATE-----

A chave privada, relacionada ao certificado compartilhado, é de responsabilidade de sua empresa e deve ser armazenada de forma segura, nunca sendo fornecida a terceiros.

2.1.1 AMBIENTE DE HOMOLOGAÇÃO

Para início dos testes, vide fornecimento do certificado digital, pode ser fornecido para cadastro o certificado gerado com o auxílio da biblioteca openssl.

Observação:

O openssl é distribuído de maneira nativa na maioria das distros Linux, caso esteja fazendo uso de algum OS sem esta biblioteca, pode ser utilizado o "Git Bash", ao qual é instalado juntamente com o "Git Client", ou em caso de OS Windows algum WSL (Windows Subsystem for Linux).

Criar no diretório de execução o arquivo "template.txt" com o seguinte conteúdo, substituindo os valores em negrito conforme necessidade, por exemplo "<estado>" tornando-se "São Paulo". Neste caso, a razão social e CNPJ estarão no *Common Name* (CN) do certificado X.509.

Quadro 02 – Arquivo template.txt para diretrizes do certificado.

```
[ req_distinguished_name ]
stateOrProvinceName      = <estado>
organizationName        = <razão social>
commonName              = <razão social> : <CNPJ>
countryName             = BR

# =====

[ v3_req ]
keyUsage                  = critical,digitalSignature
subjectKeyIdentifier      = hash
extendedKeyUsage          = clientAuth, serverAuth

[ req ]
default_bits             = 2048
default_md               = sha256
default_keyfile         = parceiro.homologacao.key.pem
```

utf8	= yes
distinguished_name	= req_distinguished_name
req_extensions	= v3_req
prompt	= no

Como boas práticas, na geração do nome do arquivo, campo “default_keyfile”, recomendam-se utilizar no lugar de “parceiro” o nome associado da sua organização.

Execute o seguinte comando para realizar a geração do par de arquivos.

Quadro 03 – Geração certificado.

Comando:

```
openssl req -new -x509 -config template.txt -nodes -out parceiro.cer.pem -days 1080
```

A expiração do certificado deve ser de no mínimo um ano. Como boas práticas, na geração do arquivo, recomendam-se utilizar o nome associado da sua organização.

Cada execução do comando, embora tenha os mesmos parâmetros, gera uma nova chave privada em um novo certificado, uma execução no mesmo diretório pode sobrescrever os arquivos já existente. Tal execução irá gerar dois arquivos: a chave privada (parceiro.homologacao.key.pem) e o certificado público (parceiro.cer.pem).

2.1.2 AMBIENTE DE PRODUÇÃO

Visando agregar maior segurança no processo, o uso de certificados emitidos por uma autoridade certificadora é recomendável, desde que na estrutura contenha os dados da razão social e CNPJ. Um certificado auto assinado também é passível, o formato do arquivo template para produção é fornecido no item [“5. Produção”](#).

2.2 ENVIO DADOS CADASTRAIS

O certificado público, e **somente o certificado público**, deve ser enviado em dois e-mails separados da seguinte maneira:

1. Certificado público zipado utilizando senha de descompactação; neste e-mail, juntamente com o certificado, deverá conter os dados:
 - Razão social e CNPJ;
 - Breve descrição/utilização da aplicação consumidora;
 - Dois contatos de referência para renovações e avisos;
2. Senha de descompactação, arquivo “.txt” como anexo;

Para o envio dos dados, o contato deve ser direcionado para a caixa “plataforma.api@bradesco.com.br”. Após a resposta com o fornecimento do ID de acesso, a utilização do ambiente de homologação Bradesco já pode ser iniciada.

3. CONSUMO APIs

Para realizar o consumo das *Open APIs*, são necessárias duas requisições/etapas.

1. Obtenção do access-token;
2. Consulta ao endpoint do serviço;

Os passos seguintes irão descrever como realizar estas duas requisições de forma manual, utilizando-se da ferramenta de requisições HTTP, *Postman*, e a biblioteca *openSSL*. Os fluxos descritos devem ser desenvolvidos para funcionar de forma automatizada nos sistemas consumidores.

Os ambientes disponíveis e sua respectiva *URLs* são:

- Homologação; <https://proxy.api.prebanco.com.br;>
- Produção; <https://openapi.bradesco.com.br;>

3.1 OBTER ACCESS-TOKEN

O token de acesso às APIs é concebido após o fornecimento de um JWS (após assinar digitalmente o JWT, ele torna-se um JWS), informando o ID da aplicação em um dos seus *claims*, tal valor é gerado no passo anterior após o compartilhamento do certificado público. No *response* da requisição realizada estará presente o token a ser utilizado na autenticação tipo *Bearer*.

3.1.1 JWT

O desenvolvedor deverá gerar um JWT de acesso e realizar a assinatura do conteúdo com sua chave privada. O JWT é uma estrutura JSON, formado por duas partes, sendo *header* e *payload*. Ao realizar a assinatura torna-se um JWS.

3.1.1.1 HEADER

Quadro 05 – Estrutura e exemplo de preenchimento do header.

Estrutura:

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

Exemplo:

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

3.1.1.2 PAYLOAD

Para este modelo de autenticação, o endpoint a ser consumido para geração do access-token é o `"/auth/server/v1.1/token"`. Por desta deve ser fornecido a URL completa no campo *audience*, acrônimo *"aud"*.

Quadro 06 – Estrutura e exemplo de preenchimento do payload.

Estrutura:

```
{
  "aud": "<URL do serviço de geração de token>",
  "sub": "<ID aplicação / client key>",
  "iat": "<data de geração atual, formato Unix timestamp (segundos)>",
  "exp": "<data de expiração, formato Unix timestamp (segundos)>",
  "jti": "<nonce – numérico de no máximo dezoito dígitos, valor sem repetição>",
  "ver": <versão>
}
```

Exemplo:

```
{
  "aud": "https://proxy.api.prebanco.com.br/auth/server/v1.1/token",
  "sub": "bc7ccf09-8a85-4be6-y67e-82bf11737994", <id cliente fornecido pelo banco>
  "iat": "1612899472", <data atual em segundos>
  "exp": "1612903071", <data atual adicionando uma hora à frente, em segundos>
  "jti": "1574094116000", <data atual em milissegundos>
  "ver": "1.1"
}
```

3.1.1.3 GERAÇÃO

Para realizar a geração do JWT, será necessário realizar o *encode* dos dois JSONs citados para *base64 url encoded*, além da remoção de todos os espaços e quebras de linha, operação conhecida como *"stringify"*.

Quadro 07 – Encode e formatação JSON para o JWT.**Comando:**

```
echo -n "$(cat <arquivo>)" | tr -d '[:space:]' | base64 | tr -d '[:space:]' | tr '+/' '-_'
```

O comando *"echo -n "\$(cat <arquivo>)"* irá realizar a leitura do arquivo, a opção *"-n"* é para que seja realizada a leitura e propagação da string sem quebra de linha no final. Por sua vez o comando *"tr -d '[:space:]'"* realiza a operação *"stringify"*, removendo caracteres especiais (espaço, *line feed* – 'LF' ou '\n'- e *carriage return* – 'CR' ou '\r'-). Já o último comando, *"tr -d '[:space:]' | tr '+/' '-_'"*, realiza a formatação para *url encoded* e facilita o *copy and paste*, devido que faz com o que o output não tenha quebras de linha.

Quadro 08 – Exemplo geração header e payload para utilização no JWT.**Exemplo (header.json):**

```
$ echo -n "$(cat header.json)" | tr -d '[:space:]' | base64 | tr -d '[:space:]' | tr '+/' '-_'  
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
```

Exemplo (payload.json):

```
$ echo -n "$(cat payload.json)" | tr -d '[:space:]' | base64 | tr -d '[:space:]' | tr '+/' '-_'
```

```
eyJhdWQiOiJodHRwczovL3Byb3h5LmFwaS5wcmViYW5jby5jb20uYnVYXV0aC9zZXJ2ZXIvdjEuMS90b2t1bilsInN1Yil6ImJjN2NjZjA5LTlhODUtNGJlNi15NjdLTgyYmYxMTczNzk5NCIsImh0dCI6IjE1NzQwOTQxMTYiLCJleHAiOiIxNTc2Njg2MTE2IiwianRpljoiMTU3NDA5NDEuXjAwMCIsInZlcil6IjEuMSJ9
```

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

Header (base64url encoded) + "." + Payload (base64url encoded)

Quadro 09 – Exemplo JWT.

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJodHRwczovL3Byb3h5LmFwaS5wcmViYW5jby5jb20uYnVYXV0aC9zZXJ2ZXIvdjEuMS90b2t1bilsInN1Yil6ImJjN2NjZjA5LTlhODUtNGJlNi15NjdLTgyYmYxMTczNzk5NCIsImh0dCI6IjE1NzQwOTQxMTYiLCJleHAiOiIxNTc2Njg2MTE2IiwianRpljoiMTU3NDA5NDEuXjAwMCIsInZlcil6IjEuMSJ9
```

3.1.2 JWS

O JWS (JSON Web Signature) é uma estrutura ao qual contém o conteúdo do JWT adicionado da assinatura gerada após o seu *digest*. Para gerar a assinatura, utilize o seguinte comando:

Quadro 10 – Exemplo geração assinatura.

Comando:

```
echo -n "$(cat <arquivo_assinar>)" | openssl dgst -sha256 -keyform pem -sign <chave_privada.pem> | base64 | tr -d '[:space:]' | tr '+/' '-_'
```

Exemplo (arquivo "jwt.txt" e chave "manual.teste.com.key.pem"):

```
$ echo -n "$(cat jwt.txt)" | openssl dgst -sha256 -keyform pem -sign manual.teste.com.key.pem | base64 | tr -d '[:space:]' | tr '+/' '-_'  
HldrEd626l8GTPdulKteaopAYBuk_Yzp9oq9_GYko6ikKIgK3ezB-  
y0yxfbzRNdUKh77c57t_re8j8EBHSyeauVRJdGfdJssU3dXH9rdFQ7yLTc7PH489oQ1x1CaC9  
HRJPOFL6Tyq5pWfQyDBL4d9b777sriAx25oq8Lvr8pKszMkCDBccfFP0cZxFN5FkoBw8ynR  
e3FSUsVV-zQTUzQ7Kr1jL3dQctBPXHJ_84qfv9tUbX6k7RmYqyOA-  
uqEC5JiqFTZzkjzGS8kbaZX0I4cDF7vKnbmhefiLlqc7PooWm-  
w65o4zqpu6iME1hqOeYkwx5JLAO_HdPR46S-lkH-Wg
```


k4LTQxZjltOTJhNC0xYzQyMWI2YmM1ZmliLCJpYXQiOiNjEwOTI0NTU1IiwiaXhwaWJoiMTYxMzUxNjU1NSIsImppOaSI6IjE2MTA5MjQ1NTUxNTMiLCJ2ZXliOiNjEifQ.4vA0Tz9bnuoVYjI4aTGLDdsSZhn6I1erX1UsLchnSND_L9nASwsHHmJi-a7oixePQnY27dQayxN2VT-6uVEfML8Rj7vS1wh2wa2y2CFDN79Dr2iSrIblimXlrPQ8gOjcSk4-8kpdyQynZuOMuZjvppMu6kUJ8uz4Pmx0B5erCN4KNV19mSuNLXhDGbbLxPhrdyOV7GRBlrEaRHFDORT5uxQDFL28C8ENUCoC17eSOMoBzPnrYLe9wal28VxHg4g2GOLvbtLXfigfmWhn5cIMiBC8YStCpb0iAUdYGucIXH1KfF94lpXloZTekS78zkgAmPvoPSutoyHcsWUlr9-4w

3.1.3 REQUISIÇÃO HTTP

Na descrição abaixo foi feito uso do *Postman*, ao qual para obtenção do token deverá conter as informações de “método” e “headers” conforme abaixo.

Quadro 14 – Requisição geração access-token.

Method POST
 URL: https://<endereço_do_ambiente>/auth/server/v1.1/token
 Body: selecionar “x-www-form-urlencoded”
 Key: grant_type = urn:ietf:params:oauth:grant-type:jwt-bearer
 Key: assertion = <JWS gerado>

O *Postman* adicionará o header “Content-Type” automaticamente, “application/x-www-form-urlencoded”.

The screenshot shows the Postman interface for a POST request. The URL is https://proxy.api.prebanco.com.br/auth/server/v1.1/token. The 'Body' tab is selected, and the format is set to 'x-www-form-urlencoded'. The body contains two key-value pairs: 'grant_type' with value 'urn:ietf:params:oauth:grant-type:jwt-bearer' and 'assertion' with value 'eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...'.

KEY	VALUE	DESCRIPTION
grant_type	urn:ietf:params:oauth:grant-type:jwt-bearer	
assertion	eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...	Inserir JWS gerado

Figura 01 – Exemplo requisição access-token, ambiente de homologação.

Quadro 15 – Retorno geração access-token.

MANUAL AOS DESENVOLVEDORES – Agosto 2021

A geração de um novo *access-token* só deve ser feita após o mesmo expirar, conforme valor recebido no retorno no atributo “*expires_in*” para valores menor do que uma hora. Por sua vez, este valor indica por quantos segundos o token é válido, lembrando que a expiração máxima é de 3600 segundos, ou seja, uma hora após a geração. O efeito colateral de gerações contínuas é se deparar com o erro abaixo.

Quadro 16 – Retorno geração contínua *access-token*.

```
{
  "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."
    }
  ]
}
```

3.2 CONSUMO ENDPOINT

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.

3.2.1 ASSINATURA

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.

3.2.1.1 FORMATAÇÃO

Nesta *string* deverão ser incluídas algumas informações que serão utilizadas na chamada do HTTP, tais como: método, URL, parâmetros, *endpoint* e inclusive o *access-token* obtido anteriormente. 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 o certificado público e a chave privada. O formato está descrito abaixo.

Quadro 17 – Formato geração assinatura consumo *endpoint*

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 abaixo 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*:



Figura 02 – Inserção método HTTP.

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

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

E no *Postman*:

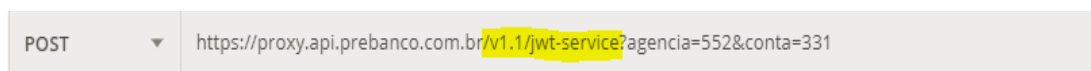


Figura 03 – Inserção *endpoint*.

- Linha 3: exemplo de parâmetros da chamada

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

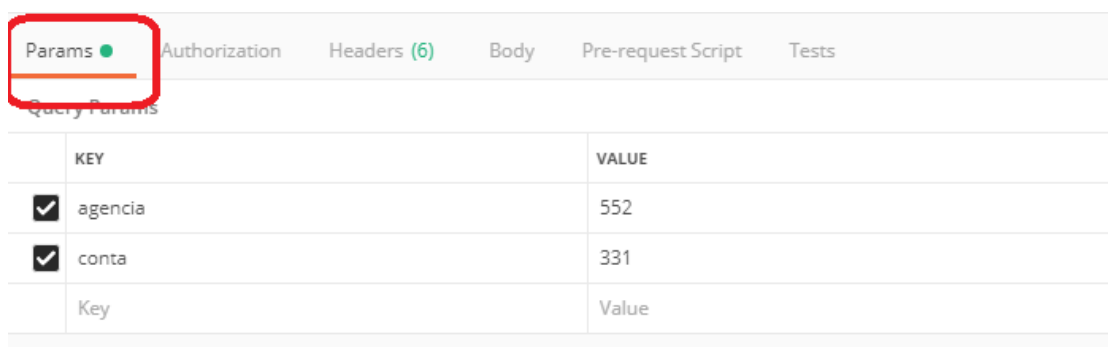
Então no *Postman* temos os parâmetros que estão sendo utilizados, podemos inserir esses valores na URI após o “?” ou na aba “*params*” do *Postman*.

Parâmetros:



Figura 04 – Inserção *query string*.

Aba *params*:

A screenshot of the Postman interface showing the 'Params' tab selected. The 'Query Params' section is visible, containing a table with two rows: 'agencia' with value '552' and 'conta' with value '331'. Both rows have a checked checkbox in the first column. Below the table is a 'Key' and 'Value' label pair.

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

Figura 05 – Inserção *query string*.

- Linha 4: *body* da chamada

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

O *body* dessa chamada deve ser “raw”, “JSON” no Postman.

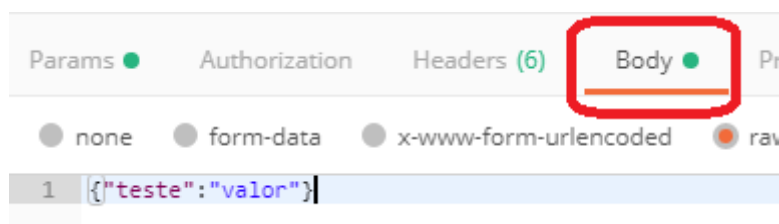


Figura 06 – Inserção JSON *body*.

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:

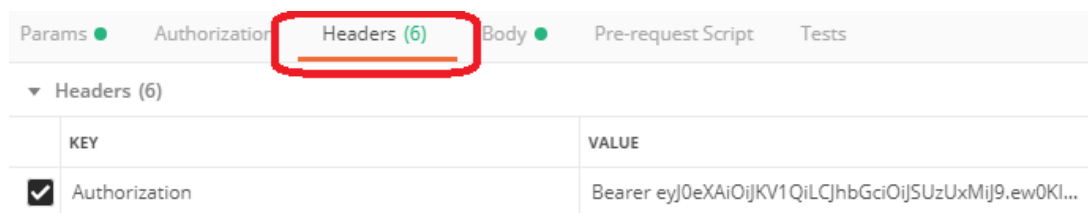


Figura 07 – Inserção *access-token*.

- 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
-------------------------------------	--------------	---------------

Figura 08 – Inserção *nonce*.

- 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 inserido no cabeçalho da chamada, sendo a sua chave "*X-Brad-Timestamp*", ficando no padrão ilustrado abaixo:


	X-Brad-Timestamp	2020-03-10T12:17:06-00:00
---	------------------	---------------------------

Figura 09 – Inserção *timestamp*.

- 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:

	X-Brad-Algorithm	SHA256
---	------------------	--------

Figura 10 – Inserção *algorithm*.

Então o arquivo para assinatura ficará da seguinte forma:

Quadro 18 – String geração assinatura consumo endpoint.

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

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, note que neste caso deixamos a linha referente ao *body* em branco.

Quadro 19 – String geração assinatura consumo *endpoint*, sem *body*.

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

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

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.

3.2.1.2 GERAÇÃO

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 está a chave privada. Execute o comando em um ambiente *Linux* para gerar assinatura. A assinatura deve estar no padrão do algoritmo utilizado, neste caso *SHA256*. O comando utilizado será o seguinte:

Quadro 20 – Exemplo geração assinatura – *digest*.

Comando:

```
echo -n "$(cat Arquivo_de_assinatura.txt)" | openssl dgst -sha256 -keyform pem -sign  
<chave_privada.pem> | base64 | tr -d '[:space:]' | tr '+/' '-_'
```

Exemplo (arquivo “request.txt” e chave “manual.teste.com.key.pem”):

```
$ echo -n "$(cat request.txt)" | openssl dgst -sha256 -keyform pem -sign  
manual.teste.com.key.pem | base64 | tr -d '[:space:]' | tr '+/' '-_'  
hAj1J6HFztwqpJaTnt4YQcUlrRuUBb-  
Uhr0N2J3TBgdXGo_UDGM3jUk4Ql6KmosNHKULfDc4SXcClpeAuYKocdD758EHwaBNuSW  
rddjq3hv2ygFbqcVocT_wf6shsJiGqcT0QnuE12QZXszalP60oloS6H4MvYXnhNJmTGZ5q6g  
cFnXyL6zpYG0Cci5SmemmWMT8f0e04-wrDD-  
XA02CrstbAovSKFc75JLrYLWcWQ6IMKMhfzmuD0Te59Wjg70-  
mTFtswJxUkehs9DNJNZS_9TFX5gPWHp5pzMx_qv5jgi-  
ugtVYL31TEck5bNMWfTF8P2hnxFaAnEQMwGx8RgVA
```

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	U1QKL2p3dC1zZXJ2aWNlcmFnZW5jaWE9NTUyJmNvb...
--	--

Figura 11 – Inserção *signature*.

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.

3.2.2 REQUISIÇÃO HTTP

Quadro 21 – Exemplo cURL gerado pelo *Postman*.

```
curl --location --request POST 'https://proxy.api.prebanco.com.br/v1.1/jwt-service' \  
--header 'Authorization: Bearer <access-token>' \  
--header 'X-Brad-Signature:  
BPaO15xM4pjA6OVOo4o0muCgb22AV1iMwZiGj4Oyzxsx5brOFzTCgFNqiVR4p3arSCU4Z  
OqqALNP1fVmrNKmfhJeCaoB3Xxbw3qlz885M5Cc3s46vJQR4vvaNZ-uS-NXP7Vj986t-  
_oKNOrCJaPrh7fsHGuN91RFnNIJ1WrFHfFdT8eMKR_qjT7_iroB4LWiB1f7smq7D30-  
OFCB398HGCBYdLbaEpK1-  
mc_yxBPc7wXW0X6WtvAPpZpDUkmmuC1d7sMXil1hJkAVdUHafqHSZvxopmBXTz1kyLT  
HF9vt23x4CKYE_ykV9thmsZnsjO431mRQLL_k_q8-8Wba_eWGg' \  
--header 'X-Brad-Nonce: 1610929465406' \  
--header 'X-Brad-Timestamp: 2021-01-17T21:24:25-00:00' \  
--header 'X-Brad-Algorithm: SHA256' \  

```

Tabela 02 – Explicação cada header utilizado.

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 resposta “API acessada com sucesso!”.

Quadro 22 – Response.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/plain;charset=UTF-8
Content-Length: 42
API acessada com sucesso!
```

4. 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;

5. PRODUÇÃO

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

- Produção; <https://openapi.bradesco.com.br;>

Criar em outro diretório de execução o arquivo “template.txt” com o seguinte conteúdo, substituindo os valores em negrito conforme necessidade, por exemplo “<estado>” tornando-se “São Paulo”.

Quadro 02 – Arquivo template.txt para diretrizes do certificado.

```
[ req_distinguished_name ]
stateOrProvinceName      = <estado>
organizationName          = <razão social>
commonName                = <razão social> : <CNPJ>
countryName               = BR

# =====

[ v3_req ]
keyUsage                  = critical,digitalSignature
subjectKeyIdentifier      = hash
extendedKeyUsage           = clientAuth, serverAuth

[ req ]
default_bits              = 2048
default_md                 = sha256
default_keyfile            = parceiro.producao.key.pem
utf8                      = yes
distinguished_name        = req_distinguished_name
req_extensions             = v3_req
prompt                    = no
```

Como boas práticas, na geração do nome do arquivo, recomendam-se utilizar no lugar de “parceiro” o nome associado da sua organização. Execute o seguinte comando para realizar a geração do par de arquivos.

Quadro 02 – Geração certificado.

Comando:

```
openssl req -new -x509 -config template.txt -nodes -out parceiro.prd.cer.pem -days 1080
```

O certificado público, e **somente o certificado público**, deve ser enviado em dois e-mails separados da seguinte maneira:

1. Certificado público zipado utilizando senha de descompactação; neste e-mail, juntamente com o certificado, deverá conter os dados:
 - Razão social e CNPJ;
 - Breve descrição/utilização da aplicação consumidora;
 - Evidência (LOG *HTTP status code 2xx*) de uma requisição à API;
 - Dois focais de referência para contato durante os processos de renovação dos certificados envolvidos;
2. Senha de descompactação, arquivo “.txt” como anexo;

Para o envio dos dados, o contato deve ser direcionado para a caixa [“plataforma.api@bradesco.com.br”](mailto:plataforma.api@bradesco.com.br).

O processo de cadastro dos certificados para produção é realizado semanalmente. O envio de todos os artefatos citados acima deve ser feito até segunda-feira às 14h. Respeitando isto, o cadastro irá ocorrer na sexta-feira da mesma semana. No caso de envios feitos após o limite estipulado (segunda-feira às 14hrs), as credenciais somente serão implantadas na próxima janela semanal, ou seja, na sexta-feira da semana seguinte.