

# Integração Cerberus

Como configurar uma aplicação client para usar o Cerberus

- [Configurar aplicação client com o Cerberus](#)
- [Autenticação via Client Credentials \(OAuth2\)](#)

# Configurar aplicação client com o Cerberus

Observações:

**Importante:** *Será disponibilizado um ambiente para os testes (Equipe TCE será responsável por enviar as informações necessárias);*

- 1 - Para realizar os testes descritos nessa documentação será necessário está conectado à uma vpn;
- 2 - A equipe TCE criará uma client para que seja possível realizar os testes;
  1. Para configurar a api para usar o Cerberus será necessário um certificado;
  2. Para facilitar a integração da aplicação angular com o Cerberus foi criada a lib ngx-cerberus (não é obrigatório o uso dessa lib, o fluxo de login pode ser implementado );

## Configurando o uso da chave pública na API

“ Obs: É indispensável essa validação para garantir que foi o token foi gerado pelo Cerberus

1. Essa chave pública pode ser armazenada dentro de um arquivo de configuração da API, como por exemplo: `src/main/resources/certs/public.txt`, essa chave é necessária para validar o token gerado.
2. A chave pública também pode ser consultada através de um endpoint disponibilizado no Cerberus, **exemplo:** `http://10.10.5.83:9999/.keys/jwks.json`.

Adicionamos a configuração da chave pública, **cert.path = certs\_dev/public.txt**

## Exemplo da configuração de validação da publicKey em Scala

1. Esse certificado é obrigatório, ele será entregue pela equipe TCE;
2. Esse certificado pode ser armazenado em um arquivo de configuração dentro da API. Ele será usado para validação da assinatura do token.
3. Para validar o certificado será necessário que na api Scala tem a validação se o certificado realmente é válido;

#### 4. Exemplo em Scala que como é feita a validação;

```
import pdi.jwt.{JwtAlgorithm, JwtBase64, JwtJson}
import pdi.jwt.algorithms.JwtAsymmetricAlgorithm
import pdi.jwt.exceptions.JwtExpirationException

/** Verifica se o token está assinado corretamente, de acordo com a chave pública do Cerberus */
protected def verifyTokenSignature(token: String, publicKey: String)(
  implicit algorithms: Seq[JwtAsymmetricAlgorithm] = JwtAlgorithms.allRSA()
): Try[Boolean] = {
  def extractTimeFrom(e: JwtExpirationException): Option[Long] = {
    /* formato do toString do [[java.time.Instant]] */
    val dateTimeRegex = """"\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.?\d{0,3}Z"""".r
    dateTimeRegex.findFirstIn(e.getMessage) match {
      case Some(time) => Some(Instant.parse(time).toEpochMilli)
      case None       => None
    }
  }

  def extractAlgorithmFrom(token: String): String =
    Try(
      Json.parse(JwtBase64.decodeString(token.split("\\.").head.mkString)) \ "alg"
    ).fold(_ => "Não informado", _.as[String])

  Try(JwtJson.validate(token, publicKey, algorithms)) match {
    case Success(_) => Success(true)
    case Failure(e: JwtExpirationException) =>
      Failure(TceJwtExpiredException(extractTimeFrom(e)))
    case Failure(e) =>
      val errorMessage : String = s"Não foi possível validar o token (criptado via ${extractAlgorithmFrom(token)})"

      val infoAboutAlgorithms = if (algorithms.length > 1) {
        s"com nenhum dos seguintes algoritmos: ${algorithms.mkString(", ")}"
      } else {
        s"utilizando o algoritmo: $algorithms."
      }

      logger.logger.error(s"$errorMessage $infoAboutAlgorithms", e)
      Failure(TceJwtInvalidTokenSignatureException())
  }
}

/** Verifica se o token está expirado (expiry > now || notBefore < now) */
protected def verifyTokenExpiration(token: String, publicKey: String)(
  implicit algorithm: Seq[JwtAsymmetricAlgorithm] = JwtAlgorithms.allRSA()
): Try[Boolean] = {
  import java.time.{Instant, ZoneOffset}
  implicit val clock: Clock = Clock.fixed(Instant.now, ZoneOffset.ofHours(-3))

  JwtJson.decode(token, publicKey, algorithm).flatMap {
    case claim if claim.isValid => Success(true)
    case claim                  => Failure(TceJwtExpiredException(claim.expiration))
  }
}

/** Verifica se o token possui a permissão especificada */
protected def verifyTokenClientId(
  validClientId: String
)(token: TceTokenClaim): Try[Boolean] = token.clientId match {
  case clientId if clientId == validClientId => Success(true)
  case _                                     => Failure(TceJwtInvalidClientId())
}

case class TceTokenClaim(clientId: String,
  id: Long,
  login: String,
  usuario: String,
  nomeCompleto: Option[String] = None,
  email: Option[String] = None,
  cpf: Option[String] = None,
  tipoUsuarioId: Option[Long] = None,
  tipoUsuario: Option[String] = None,
  sistemaId: Option[Long] = None,
  sistema: Option[String] = None,
  authorities: Seq[String] = Seq.empty[String],
  scope: Option[Seq[String]] = None,
  ip: Option[String] = None,
  jti: Option[String] = None,
)
```

tokenVerification.scala 4 KB ↓ <>

Obs: Visando facilitar a integração da aplicação angular com o cerberus desenvolvemos a lib ngx-cerberus

## Configurando o uso da lib ngx-cerberus em uma aplicação angular

Essa lib é responsável por fazer o gerenciamento do usuário na aplicação;

1. É necessário instalar a lib ngx-cerberus, para instalar essa lib é necessário ter na raiz do projeto um arquivo .npmrc com a seguinte configuração:

```
@tcepb:registry=https://gitlab.tce.pb.gov.br/api/v4/projects/366/packages/npm/  
//gitlab.tce.pb.gov.br/api/v4/projects/366/packages/npm/:_authToken=TOKEN
```

(para ter acesso ao repositório), após a criação desse arquivo, executar o seguinte comando **npm install @tcepb/ngx-cerberus@0.5.0;**

**Importante:** A lib ngx-cerberus utiliza algumas outras dependências, dentre elas, é utilizada a @auth0/angular-jwt;

### 2. Configurando o arquivo environment.ts

1. Será necessário adicionar as configurações da url da aplicação Cerberus, o clientID, o clientSecret; (Os valores para essas informações serão passados pela equipe do TCE);

#### Exemplo do arquivo environment.ts

```
export const environment = {  
  production: false,  
  name: EnvironmentNameEnum.DESENVOLVIMENTO,  
  auth: {  
    providerURL: 'http://localhost:9999',  
    clientID: 'diligencia',  
    clientSecret: 'SucGYKMKj1vqNDdvSNrd56U5ULZX1eNrIYckzFBEdhwCwhPLRA7pvX7Zjrg9eTFzN0RYLfZ019pZtUdJmPihKUMAT741BxvYI',  
  },  
  apiURL: 'http://localhost:9005',  
  apiVersion: 'v1',  
  version: 'DEV_2.0.2-SNAPSHOT',  
  whitelistedDomains: ['localhost:9005', 'localhost:9999']  
};
```

### 3. Configurando os parâmetros que serão necessários para o NgxCerberus

1. No arquivo app.module.ts será necessário realizar as configurações para o uso da lib ngx-cerberus;
2. import HTTP\_INTERCEPTORS, JwtInterceptor, JwtModule, getToken, NgxCerberusHttpInterceptor, NgxCerberusModule, NgxCerberusService, environment(no exemplo configuramos algumas variáveis necessárias para JwtModule nesse arquivo );
- 3.

### Exemplo das importações:

```
import { HTTP_INTERCEPTORS, HttpClientModule } from '@angular/common/http';
import { JwtInterceptor, JwtModule } from '@auth0/angular-jwt';
import { environment } from 'src/environments/environment';
import { getToken, NgxCerberusHttpInterceptor, NgxCerberusModule,
NgxCerberusService } from '@tcepb/ngx-cerberus';
```

4. Configurando os parâmetros necessários para o módulo `JwtModule`; Será usado o `.forRoot` método de `JwtModule` para fornecer um objeto de configuração com os seguintes atributos:

Exemplo:

```
const APP_MODULES = [
  JwtModule.forRoot({
    config: {
      tokenGetter: getToken,
      authScheme: 'Bearer ',
      headerName: 'Authorization',
      skipWhenExpired: true,
      whitelistedDomains: ['localhost:9005', 'localhost:9999'],
      blacklistedRoutes: [
        environment.auth.providerURL + '/cerberus/oauth/authorize',
        environment.auth.providerURL + '/cerberus/oauth/token',
      ],
    },
  }),
];
```

Será necessário informar um objeto de configuração:

- `tokenGetter`: esta função é usada para personalizar como `JwtModule` e obter o token de acesso JWT do armazenamento local.
- `whitelistedDomains`: nesta matriz, você pode adicionar qualquer domínio que tenha permissão para receber o JWT como APIs públicas.
- `blacklistedRoutes`: nesta matriz, você pode adicionar rotas que não têm permissão para receber o token JWT.

5. Configurando o objeto necessário para o módulo `NgxCerberusModule`:  
Será necessário informar os seguintes valores para o objeto: `clientID`, `clientSecret`, `providerURL` (essas informações serão passadas pela equipe TCE) ;  
Exemplo:

```
/** TCE MODULES */
const ngxCerberusConfig = {
  clientID: environment.auth.clientID,
  clientSecret: environment.auth.clientSecret,
  providerURI: environment.auth.providerURL,
};
const TCE_MODULES = [NgxCerberusModule.forRoot(ngxCerberusConfig)];
```

6. Adicionar ao array de **imports** o módulo `JwtModule`, `NgxCerberusModule`;
- - Adicionar ao array de **providers** o `HTTP_INTERCEPTORS` utilizando o `JwtInterceptor`, `NgxCerberusHttpInterceptor` e o `NgxCerberusService`;

Exemplo:

```
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    HttpClientModule,
    ...APP_MODULES,
    AppRoutingModule,
    TCE_MODULES,
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: NgxCerberusHttpInterceptor,
      multi: true,
    },
    [NgxCerberusService],
  ],
})
```

Ao concluir essas configurações e executar o projeto, se tudo deu certo, você deverá ser redirecionado para tela de login do Cerberus;



## **Repositório de exemplo de uma aplicação angular utilizando a lib ngx-cerberus: xxxxxxxx**

### Integrando um aplicativo mobile ao Cerberus

#### Realizando a requisição para obter um code

- **Observação: os dados necessários como endpoint para obtenção de code, token, clientId, clientSecret, usuário e senha de testes deverão ser passados pela equipe de desenvolvedores do tce.**

Para obter um Authorization code do Cerberus será necessário realizar uma requisição para o endpoint <http://10.10.5.83:9999/oauth/authorize>, essa requisição precisa ser apenas um redirect para esse endpoint;

Exemplo da requisição:

```
`http://10.10.5.83:9999/oauth/authorize?response_type=code&client_id=appCarteiraFuncional&redirect_uri=http://127.0.0.1:8100`
```

, essa requisição irá abrir a tela de login do Cerberus com as informações do client que está sendo passado como parâmetro;

Após efetuar o login no Cerberus será retornado juntamente com a url de retorno(sua url) o code. Com o code de acesso agora é possível obter um token jwt;

#### **Trocando o code recebido por um token jwt**

Para obter o token jwt será necessário enviar uma requisição do tipo POST para o Cerberus no endpoint <http://10.10.5.83:9999/oauth/token> informando os parâmetros necessários;

Parâmetros que deverão ser passados: **grant\_type**(do tipo authorization\_code), **client\_id**(fornecido pela equipe do tce), **code**(recebido na requisição para o obter um code), **redirect\_uri**(sua url que receberá o retorno do Cerberus);

Exemplo da requisição:

```
const params = new HttpParams()
.set("grant_type", "authorization_code")
.set("client_id", environment.clientId)
.set("code", code)
.set("redirect_uri", "http://127.0.0.1:8100");
const headers = new HttpHeaders()
.set("Content-type", "application/json; charset=utf-8")
.set(
  "Authorization",
  "Basic " + btoa(environment.clientId + ":" + environment.clientSecret)
);
`http://10.10.5.83:9999/oauth/token?grant_type=authorization_code&client_id=appCarteiraFuncional&code=CODE_RECEBIDO_AO_EFETUAR_LOGIN_NO_CERBERUS&redirect_uri=http://127.0.0.1:8100`;
```

# Autenticação via Client Credentials (OAuth2)

Observações:

Esse fluxo é utilizado por sistemas, não envolve nenhum login de um usuário comum. Em vez disso, a aplicação usa suas próprias credenciais (`client_id` e `client_secret`) para se autenticar e obter um token de acesso.

**Importante:** Para a realização da integração será disponibilizado pela equipe do TCE um **clientId** e um **Secret**;

Antes de começar, você precisa receber da equipe TCE os seguintes dados:

- **Client ID** (Identificador do cliente)
- **Client Secret** (Senha secreta do cliente)
- **Token URL** (<https://login-teste.tce.pb.gov.br/oauth/token>)

## Etapas para obter o token

Para obter o token é necessário fazer uma requisição para provedor de autenticação do TCE (Cerberus), para isso, é preciso utilizar os seguintes dados:

Monte uma requisição HTTP `POST` para o `token_url`, com os seguintes parâmetros:

```
POST https://login-teste.tce.pb.gov.br/oauth/token
```

```
Headers: Content-Type: application/x-www-form-urlencoded  
Authorization: Basic <base64(seu_client_id:seu_client_secret)>
```

## Exemplos de requisição:

### Utilizando o Curl

Será necessário codificar o `clientId` e o `secret` em `base64` para ser passado no `Authorization`

Exemplo de como codificar em `base64` usando o terminal

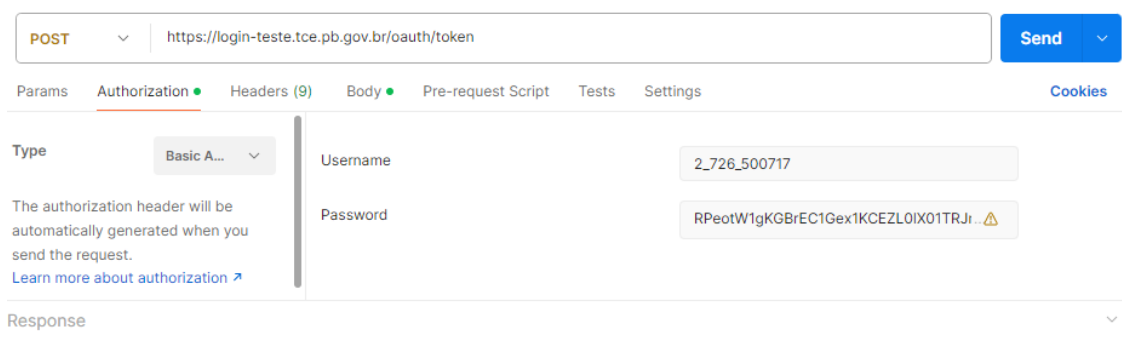
```
echo -n "SEU_CLIENT_ID:SEU_SECRET" | base64
```

```
curl -X POST https://login-teste.tce.pb.gov.br/oauth/token -H "Content-Type: application/x-www-form-urlencoded" -H "Authorization: Basic SEU_CLIENT_ID e SEU_SECRET codificado em base64" -d "grant_type=client_credentials"
```

## Utilizando o Postman

### Aba Authorization

É necessário definir o tipo de Authorization como **Basic Auth**, informar o Username (o clientId) e o Password (o secret);



The screenshot shows the Postman interface with the Authorization tab selected. The URL is `https://login-teste.tce.pb.gov.br/oauth/token` and the method is `POST`. The Authorization Type is set to `Basic Auth`. The Username field contains `2_726_500717` and the Password field contains `RPeotW1gKGBrEC1Gex1KCEZL0IX01TRJr.` with a warning icon. The Response section is currently empty.

### Aba Body

É necessário definir uma key **grant\_type** com o valor `client_credentials`;



The screenshot shows the Postman interface with the Body tab selected. The URL is `https://login-teste.tce.pb.gov.br/oauth/token` and the method is `POST`. The Body type is set to `form-data`. A table below shows the form data:

Key	Value	...	Bulk Edit
<input checked="" type="checkbox"/> grant_type	client_credentials		
Key	Value		

The Response section is currently empty.

## Exemplo de retorno caso tudo ocorra com sucesso

