Autoriza - Referência

Autoriza

SUPDG/DGGDI/DGGD3

SERPRO

Índice

1. Introdução

O controle de acesso, que rege "*quem pode fazer o quê*" em uma aplicação, é dividido em dois grandes assuntos: **autenticação** e **autorização**.

A **autenticação** identifica o usuário, solicitando suas credenciais, e garantindo que ele é *realmente* quem ele está dizendo ser. Quando falarmos em **autenticação**, é natural que tratemos de termos como *login, logout, sessão, tokens de acesso, etc*. A **autenticação** responde à pergunta "*quem?*" no controle de acesso.

Já a **autorização** estabelece que privilégios o usuário possui, uma vez que ele foi autenticado. Ela garante que o usuário só consegue realizar as operações para as quais ele possui privilégios, nos recursos da aplicação que são acessíveis para ele. Quando falarmos em **autorização**, é natural que tratemos de termos como *perfil, transação, permissão, acesso negado, etc*. A **autorização** responde à pergunta "*o quê?*" no controle de acesso.

1.1 O que é o Autoriza?

O produto **Autoriza** visa prover aos sistemas do Governo Federal uma plataforma única para realização da **autorização** de acesso em diferentes aplicações de forma centralizada, bem como fornecer aos gestores das aplicações um ponto único para a administração e o controle das regras de acesso aos recursos disponíveis.

O **Autoriza** está integrado ao **Acesso gov.br** para ser utilizado como sistema de **autorização** das aplicações que realizam a **autenticação** pelo **Acesso gov.br**.

Durante a autorização de acesso do usuário, o **Autoriza** coleta e manipula as informações necessárias para **liberar**:

Autoriza liberando o acesso a um recurso da aplicação

ou **negar** o acesso a um recurso da aplicação:

Autoriza negando o acesso a um recurso da aplicação

1.1.1 Serviços oferecidos

- ✓ API para um sistema recuperar os perfis e transações do usuário logado no **Acesso gov.br**
- ✓ Aplicação das políticas de acesso a um sistema através do consumo de serviços
- ✓ Biblioteca java para consumo, resolução e aplicação das políticas de acesso
- ✓ Transformação de características do usuário, definidas através do consumo de serviços de terceiros, em perfis e transações
- ✓ Interface visual para cadastro de perfis, transações e atribuição delas ao usuário, em um cenário multisistema e multicliente
- ✓ Auditoria das manutenções nos dados relacionados ao controle de acesso
- ✓ API para um sistema manter os dados relacionados ao controle de acesso, ou incluir mudanças relacionadas ao controle de acesso em algum dos seus fluxos de negócio

1.1.2 Quem usa?

Sistemas de Governo que realizam a autenticação pelo **Acesso gov.br**.

1.1.3 Como funciona?

> **Formato da comunicação**
>
> Todas as requisições para as **APIs do Autoriza** são HTTPS, através de chamadas a *web services REST*, que consomem e produzem *dados JSON*.

1. O sistema integrado ao **Autoriza** realiza a autenticação utilizando o **Acesso gov.br**. Da autenticação, duas informações são guardadas pelo sistema: o CPF e o *token de acesso* do usuário
2. Utilizando o CPF e o *token de acesso*, o sistema aciona o **Autoriza** para validar o acesso do usuário

Esse roteiro de funcionamento pode variar dependendo do modelo de integração com o **Autoriza** adotado pelo sistema, conforme será visto na seção específica.

1.1.4 Como está dividido o guia?

1. Conceitos
2. Modelos de integração
3. Como aderir ao uso do Autoriza
4. Boas práticas de uso do Autoriza
5. Documentação OpenAPI dos serviços de autorização e validação
6. Release Notes

2. Conceitos

O **Autoriza** implementa primariamente um controle de **autorização** baseado nos *perfis* e *transações* de um usuário (RBAC).

Cada aplicação parceira do **Autoriza** precisa ser um *sistema* cadastrado no **Acesso gov.br**. O *sistema* deve possuir ao menos um *perfil* de acesso, e pode ter *transações* associadas. Cada *transação* deve estar vinculada a, no mínimo, um *perfil*.

Opcionalmente, um *sistema* pode definir suas *políticas de acesso* no **Autoriza**. Para sistemas que utilizam a feature *Tag* , podem definir uma política específica relacionada a uma *Tag* cadastrada para o *sistema*. Essas políticas são definidas no formato PML, no padrão:

```
nome dos perfis,nome das transações,identificador do recurso,verbo HTTP
```

confome os exemplos:

```
AUDITOR|EQUIPE TECNICA,*,/private/protocolo,GET
*,GESTAO PROTOCOLOS,/private/protocolo,POST
```

A primeira linha permite que um *usuário* com o *perfil* **AUDITOR** ou **EQUIPE TECNICA** realize uma requisição **GET** para o recurso **protocolo**. A segunda permite que um *usuário* com acesso a *transação* **GESTAO PROTOCOLO** realize uma requisição **POST** para o recurso **protocolo**.

Os *perfis* são concedidos para um *usuário* através de uma *atribuição* de *perfis*.

O controle de acesso é realizado aplicando as:

políticas de acesso do *sistema* ou do *sistema* e *tag*

uma vez que são recuperados:

os *perfis* e *transações* que o

usuário possui,

para definir se o *usuário*

pode ou

não acessar o

recurso da aplicação que está solicitando.

Para sistemas que possuem a configuração da features Tag, caso o parâmetro Tag seja informado, serão considerados os perfis associados ao valor de parametro Tag informado, assim como as avaliações de acesso serão baseadas na política específica da Tag informada.

No processo de avaliação de uma política de acesso é considerada a credencial de acesso do usuário logado no sistema **Acesso gov.br**.
Para um Perfil concedido ao usuário ou para uma Transação vinculada a um Perfil, pode ser configurado o indicador de 'Obrigatoriedade de uso de Certificado Digital'. Caso este indicador esteja marcado, o Autoriza irá restringir ou permitir o acesso:

1) Caso o usuário tenha logado com Certificado Digital no **Acesso gov.br** , todos os perfis e transações atribuídos ao usuário no sistema requisitado serão retornadas, independentes de exigirem ou não Certificado Digital.

2) Caso o usuário tenha logado com credencial de acesso diferente de Certificado Digital (ex.: usuário e senha), serão retornadas apenas os perfis e transações atribuídas ao usuário
no sistema que não exigem uso de Certificado Digital.

O **Autoriza** é uma solução *multi-tenancy*, atende a diversos clientes e sistemas e, como tal, precisa suprir as diversas necessidades particulares de cada sistema em termos de controle de **autorização** de acesso. Para isso, ele oferece uma série de configurações opcionais que são selecionadas individualmente para cada *sistema*, formando um conjunto exclusivo para ele. Essas configurações opcionais são as *features* e *características de usuário*.

2.1 Glossário

Definição de termos e siglas usados na documentação.

Termo	Descrição
JWT	JSON Web Tokens → Formato baseado em JSON para troca de informações entre aplicações de forma segura.
PML	PERM Modeling Language → Linguagem específica de domínio capaz de expressar regras em vários modelos de controle de acesso.
RBAC	Role Based Access Control → Controle de acesso baseado nos papéis/funções de um usuário.
ABAC	Attribute-based access control → Controle de acesso baseado em atributos do usuário e/ou ambiente.
RSA	Rivest-Shamir-Adleman → Sistema de criptografia assimétrica usado para transmissão segura de dados.
XACML	eXtensible Access Control Markup Language → Padrão que define uma linguagem, uma arquitetura e um modelo de processamento para controle de acesso a uma aplicação.
Aplicação Parceira	Sistema que utiliza o **Autoriza** no controle de acesso.
Provider / Adapter	Biblioteca do **Autoriza** embarcada na aplicação parceira para realização da **autorização**.
Gestor de Acesso	Usuário que tem permissão para atribuir perfis, de alguma aplicação parceira, para outros usuários na interface administrativa do **Autoriza**.
Gestor de Sistema	Usuário que tem permissão para criar perfis e transações, para alguma aplicação parceira, na interface administrativa do **Autoriza**.

2.2 Conceitos do Domínio

O diagrama abaixo ilustra o relacionamento entre as entidades básicas do modelo de domínio do **Autoriza**.

Representação básica do domínio

Conceito	Descrição
Usuário	Cidadão brasileiro com CPF cadastrado no **Acesso gov.br**.
Sistema	Sistema de governo integrado ao **Acesso gov.br**.
Perfil	Possível função de um usuário na aplicação cliente. Os perfis estão diretamente relacionados às **personas** da aplicação cliente, ou **atores** dos casos de uso da aplicação cliente.
Perfil Agregador	Especialização de Perfil, que possui perfis associados como filhos.
Transação	Possível permissão de um usuário na aplicação cliente. As transações são os elementos que permitem a divisão dos acessos da aplicação por conjunto de funcionalidades, e a posterior distribuição dessas possibilidades de acesso entre os perfis. Pode ser configurada para uma transação se exige login com Certificado Digital no **Acesso gov.br**.
Política de Acesso	Elemento que discrimina quais perfis e/ou transações o usuário precisa possuir para ter acesso a um determinado recurso da aplicação. É expressa em PML.
Atribuição	Registra a concessão de um determinado conjunto de perfis a um usuário em um sistema. É recuperando as atribuições de um usuário que o **Autoriza** determina se ele pode ou não acessar determinado recurso.
Tag	Etiqueta de classificação para perfis e políticas de acesso para facilitar a governança dos dados do domínio de autorização da aplicação cliente.

2.3 Features

Além do controle de acesso básico implementando o RBAC, o **Autoriza** oferece uma série de customizações possíveis para uma aplicação parceira. Essas customizações, chamadas de *features*, são definidas em termos de novas informações que precisam ser fornecidas por um gestor de acesso ao atribuir um novo conjunto de *perfis* para um *usuário*.

As novas informações correspondem a novos campos que são agregados ao conjunto de *perfis* da *atribuição*, e passam a ser tratados em todos os fluxos do controle de acesso, de uma forma tal que possibilita, à aplicação parceira, a utilização de qualquer uma dessas informações como critério de filtragem para o controle de acesso.

O diagrama abaixo ilustra o relacionamento entre *sistema* e *feature* no modelo de domínio do **Autoriza**.

Representação domínio features

As *features* disponíveis para uso na configuração das aplicações parceiras são as listadas a seguir:

Feature	Descrição	Campo correspondente na atribuição	Tipo do campo
Certificado Digital	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** com certificado digital.	exigeCertificadoDigital	Booleano
Vigência	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** durante o período definido por data inicial e data final,respeitando o formato AAAA/MM/DD	dataInicioVigencia e dataFimVigencia	Texto
Horário	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** durante o intervalo de tempo diário definidos pela hora inicial e final, respeitando o formato HH:MM	horaInicio e horaFim	Texto
Dias da Semana	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** em determinados dias da semana. Os dias são respectivamente indicados por uma sequência de sete dígitos numéricos, iniciando pelo Domingo. O valores aceitos são 1 para permissão de acesso e 0 para negação.	diasDaSemana	Texto
Órgão SIAPE	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** em um determinado órgão. O código do órgão pode ser utilizado como critério de filtragem das atribuições de perfis do usuário.	cdOrgaoSiape	Numérico
UORG SIAPE	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** em uma determinada UORG. O código da Uorg pode ser utilizado como critério de filtragem das atribuições de perfis do usuário.	cdUorgSiape	Numérico
UPAG SIAPE	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** em uma determinada UPAG. O código da UPAG pode ser utilizado como critério de filtragem das atribuições de perfis do usuário.	cdUpagSiape	Numérico
Nível de Acesso	Define que um determinado conjunto de *perfis* só pode ser utilizado por um usuário logado no **Acesso gov.br** em um determinado nível de acesso. O nome do nível de acesso pode ser utilizado como critério de filtragem das atribuições de perfis do usuário.	nivelAcesso	Texto
Configurável	Permite definir um campo texto arbitrário, que satisfaça a uma expressão regular também customizável, a ser solicitado durante a *atribuição* de *perfis* para um *usuário*.	Definido na configuração da feature para o sistema.	Texto
Tag	Identifica um agrupamento de perfis na aplicação cliente. As tags são os elementos que marcam os perfis para determinado agrupamento ou identificação de conceito no sistema. Define que um determinado conjunto de *perfis* pode ser submetido a uma política definida para uma determinada tag durante a avaliação de acesso de um usuário logado no **Acesso gov.br** .	-	-

Feature	Descrição	Campo correspondente na atribuição	Tipo do campo
Habilitação	Permite que uma aplicação cliente repita perfis atribuídos para um mesmo usuário, desde que as atribuições sejam para valores diferentes das demais features da aplicação cliente, conforme configuração realizada na interface administrativa do **Autoriza** para definir o critério de unicidade a ser utilizado na verificação de duplicidade de atribuição de perfil.	-	-
Perfil Agregador	Permite que uma aplicação cliente crie conjuntos de perfis, através de uma relação pai-filhos (perfil agregador - perfis simples), permitindo que um gestor de sistema adicione (ou remova) perfis de vários usuários num mesmo momento, ao manipular o conjunto de perfis sob um perfil pai atribuído a vários usuários.	-	-

2.3.1 Exemplo do uso de features

O conjunto de *features* que é utilizado é particular para cada *sistema*, permitindo que uma aplicação parceira mais simples (que chamaremos de "Aplicação Simples") utilize apenas o RBAC básico, e uma outra aplicação com um controle de acesso mais complexo (que chamaremos de "Aplicação Complexa") utilize o RBAC, a *feature* **Certificado Digital** e ainda uma *feature* **Configurável** nomeada **Protocolo**, que aceita valores de 10 dígitos númericos, por exemplo.

Para a "Aplicação Simples", os *perfis* e *transações* que um *usuário* possui são retornados pelo **Autoriza** sem campo adicional de informação, permitindo uma aplicação direta do RBAC.

Para a "Aplicação Complexa", os *perfis* e *transações* são retornados associados aos valores dos campos correspondentes às *features*:

- exigeCertificadoDigital
- protocolo

Então é possível, ainda que não seja necessário, que a aplicação parceira parametrize as funções de controle de acesso do **Autoriza** para elas só considerarem os conjuntos de *perfis* atribuídos cujo valor do **protocolo** é "*2368947512*", por exemplo. Adicionalmente, o **Autoriza** não considerará os conjuntos de perfis com valor do **exigeCertificadoDigital** como "*true*" caso o *usuário* **não** esteja logado com certificado digital de pessoa física no **Acesso gov.br**.

Para uma outra aplicação, que utilize as configurações para **vigência**, **horário**, **dias da semana**, **nível de acesso**, **órgão SIAPE**, **uorg SIAPE** e **upag SIAPE**, os perfis e transações são retornados associados aos valores dos campos correspondentes às *features*:

- exigeCertificadoDigital
- dataInicioVigencia e dataFimVigencia (vigência)
- horaInicio e horaFim (horário)
- diasDaSemana
- cdOrgaoSiape
- cdUorgSiape
- cdUpagSiape
- nivelAcesso

O **Autoriza** considerará os perfis cujos valores data, horário, dia da semana, nível de acesso, código de órgão do SIAPE, código de UORG do SIAPE e código de UPAG do SIAPE da realização do login estejam, respectivamente, dentro dos valores configurados para os campos das features correspodentes.

2.4 Tags

As *tags* são etiquetas de classificação para segmentar os perfis e políticas de acesso de um sistema.

O diagrama abaixo ilustra o relacionamento entre *sistema*, *políticas de acesso*, *perfil* e *tag* no modelo de domínio do **Autoriza**.

Representação domínio tags

Quando um sistema utiliza a *feature* **Tag**, é possível associar etiquetas aos perfis e políticas de acesso da aplicação cliente, de tal modo a permitir que, em um dado momento, durante o processo de autorização de acesso, o tráfego de informação de autorização de acesso entre a aplicação cliente e o **Autoriza** trata apenas de um subconjunto das informações de autorização cadastradas para a aplicação e o usuário.

O uso das *tags* permite uma sensível diminuição na quantidade de informação que deve ser trafegada entre as aplicações durante o processo de autorização, conforme figura a seguir.

Acesso com adapter utilizando tags

2.5 Habilitação

A feature habilitação influencia o controle da atribuição de perfis realizada pelo **Autoriza**:

- Caso a aplicação parceira **não** utilize a feature, um gestor de acesso pode atribuir um perfil para um usuário apenas uma vez.
- Caso a aplicação cliente **utilize** a feature, um gestor de acesso pode atribuir um perfil para um usuário mais de uma vez, seguindo os critérios de unicidade definidos na configuração da aplicação cliente.

Ela também permite que um sistema trate isoladamente uma atribuição de perfis durante o processo de autorização. Ainda que o usuário possua várias atribuições de perfis, a aplicação cliente pode informar que deseja tratar uma atribuição específica durante o processo de autorização, limitando os perfis do usuário que serão utilizados para verificar o acesso.

Acesso com adapter utilizando habilitacao

2.6 Perfil Agregador

Os *perfis agregadores* são perfis especiais, que possuem um conjunto de perfis filhos associados.

O diagrama abaixo ilustra o relacionamento entre *sistema*, *perfil*, *perfil agregador* e *transacao* no modelo de domínio do **Autoriza**.

Representação domínio perfil agregador

Quando um sistema utiliza a *feature* **Perfil Agregador**, é possível construir conjuntos de perfis, que ficarão sob o perfil agregador, de tal modo a permitir que, em um dado momento, o gestor de sistema adicione ou remova perfis de vários usuários sem ter que manipular individualmente cada atribuição.

Durante o processo de autorização, todos os perfis filhos (e suas respectivas transações) dos perfis agregadores atribuídos ao usuário são também considerados como perfis e transações que o usuário possui.

2.7 Características de Usuário

O **Autoriza** também tem sua modalidade de ABAC, com o controle de acesso baseado em determinadas *características de usuário*. Essas *características* fazem parte do modelo de negócio e estão sob a gestão de outros sistemas, dos quais o **Autoriza** consome a informação.

Uma aplicação parceira que deseja utilizar determinada *característica* define que *perfis* devem ser atribuídos ao *usuário* caso ele possua a *característica*.

O **Autoriza**, após perguntar ao sistema "dono" da informação se o *usuário* possui uma determinada *característica*, utiliza essa resposta para compor os *perfis* e *transações* que o *usuário* possui.

O diagrama abaixo ilustra o relacionamento entre *sistema*, *característica de usuário* e *perfil* no modelo de domínio do **Autoriza**.

Representação domínio características

As *características* disponíveis para uso na configuração das aplicações parceiras são as listadas a seguir:

Característica	Descrição	Sistema que resolve a característica
Ex-servidor	Ex-servidor da administração pública da União	SIGEPE
Servidor	Servidor da administração pública da União	SIGAC
Aposentado	Aposentado da administração pública da União	SIGAC
Pensionista	Beneficiário de pensão de um servidor da administração pública da União	SIGAC
Chefe	Servidor com cargo de autoridade (chefia) em algum órgão da administração pública da União	SIGAC
Substituto	Servidor com cargo de autoridade (substituto de chefia) em algum órgão da administração pública da União	SIGAC

2.7.1 Exemplo do uso de características de usuário

A aplicação "Aplicação exemplo" tem a *característica* **Ex-servidor** configurada para garantir o *perfil* **CLIENTE DADOS FINANCEIROS** para aquele *usuário* que a possui.

E possui uma página cujo acesso é restrito apenas a *usuários* com o *perfil* **CLIENTE DADOS FINANCEIROS**, conforme a *política de acesso*:

```
CLIENTE DADOS FINANCEIROS,*,/private/rendimentos,*
```

Então o controle de acesso realizado pelo **Autoriza** para a aplicação parceira seria similar ao ilustrado no diagrama abaixo.

Exemplo de resolução de características

Apesar do *usuário* não possuir o *perfil* **CLIENTE DADOS FINANCEIROS** através de uma *atribuição* direta no **Autoriza**, ele acaba recebendo o *perfil* pois o **SIGEPE** responde positivamente quando o **Autoriza** pergunta se o CPF possui a *característica* **Ex-servidor**.

3. Modelos de Integração

3.1 Modelos de Integração

Há três possíveis modelos de integração que podem ser utilizados por uma aplicação parceira.

Esses modelos podem ser mesclados conforme a necessidade particular da aplicação.

Em todos eles, o **Autoriza** funciona como um cadastro centralizado das informações de **autorização**, mas cada um possui seus benefícios e cenários indicados de uso, conforme seções específicas linkadas abaixo.

Os modelos disponíveis são:

Consumo de informações de autorização → Cadastro centralizado de informações de **autorização** + fonte de dados para o processo de **autorização**

Integração com o provider → Cadastro centralizado de informações de **autorização** + fonte de dados para o processo de **autorização** + provedor da *engine* para aplicação de *políticas de acesso*, com um ponto único de recuperação das *políticas de acesso*

Aplicação remota das políticas de acesso → Cadastro centralizado de informações de **autorização** + aplicação remota das *políticas de acesso*, que inclui a auditoria das autorizações de acesso no **Autoriza**

3.2 Consumo de Informações de Autorização

3.2.1 Consumo das Informações de Autorização

O **Autoriza** pode atuar como um mero provedor das informações necessárias para **autorização** de acesso.

A aplicação parceira fica responsável por de fato aplicar essas informações no controle de acesso. Ou seja, é a aplicação parceira que aplica as *políticas de acesso* ao *sistema*, que são desconhecidas pelo **Autoriza**.

Nesse cenário, o sistema utilizará os serviços da API de autorização, conforme detalhado no exemplo de uso.

Conforme falado no início, o *sistema* precisa resolver a autenticação do *usuário* antes de acionar o **Autoriza**. Ou seja, se o *usuário* ainda não fez o login, ele precisa fazer.

O fluxo de troca de mensagens de controle de acesso para um usuário **ainda não logado** na aplicação está representado no diagrama a seguir.

Autoriza como fornecedor dos dados de autorização para um usuário ainda não autenticado

Caso o *usuário* já esteja logado na aplicação, ela já pode acionar diretamente os serviços do **Autoriza**.

O fluxo de troca de mensagens de controle de acesso para um usuário **já logado** na aplicação está representado no diagrama a seguir.

Autoriza como fornecedor dos dados de autorização para um usuário já autenticado

> **Guardar informações de autorização em cache ou token de aplicação**
>
> Só é necessário consultar os dados do **Autoriza** uma vez por sessão do usuário.
> Guarde - em memória, numa cache externalizada ou em um token de aplicação - as informações assim que você recuperá-las do **Autoriza**.
>
> Esse procedimento melhora a experiência do usuário uma vez que a **autorização** é um processo contínuo, que deve ser revalidado sempre que um usuário solicita um novo recurso (clica num botão ou num link, por exemplo). Fazer uma chamada remota para o **Autoriza** a cada interação do usuário com sua aplicação vai prejudicar demais a performance do seu sistema para o usuário.

> **Auditoria**
>
> Ao adotar esse modelo, a aplicação parceira fica responsável por implementar a auditoria dos acessos realizados por um usuário.

O que é preciso cadastrar no Autoriza?

Para possibilitar essa integração, é preciso que seja realizado, para o *sistema*, o cadastro das seguintes entidades no **Autoriza**:

- Sistema
- Tags (caso o sistema possua a feature)
- Perfis
- Transações (opcional)
- Atribuições de perfis a usuários
- Features e características do usuário (opcional)

Benefícios

- Possibilita ao gestor a administração centralizada da criação de perfis e transações e atribuição desses aos usuários
- Possibilita ao gestor o rastreamento das mudanças realizadas nos cadastros de perfis, transações e atribuições através da auditoria
- Delega para o **Autoriza** a manutenção, evolução e suporte dos cadastros relacionados à **autorização**, possibilitando à aplicação parceira um maior foco nas demandas diretamente relacionadas aos seus fluxos de negócio
- Permite o uso integrado do **Autoriza** com bibliotecas de mercado que implementam o RBAC, diminuindo o tempo de implementação do controle de **autorização**

Limitações

- A governança do controle de acesso pelo gestor fica limitada
- A aplicação parceira fica responsável por aplicar as políticas de acesso
- Eventuais mudanças nas políticas devem gerar um novo *release* da aplicação, demandando esforço de implementação pela equipe de desenvolvimento da aplicação

Cenários indicados para uso

O uso desse modelo de integração, por ser o de curva de adoção mais suave, é indicado para aplicações "legadas", que já possuem um controle de acesso baseado em RBAC, e uma lógica de aplicação das políticas espalhada pelo código da aplicação. Nesse caso, o **Autoriza** funciona como um susbtituto do antigo provedor de informação de **autorização**.

3.2.2 Exemplo de Uso para o Consumo das Informações de Autorização

A API utilizada é a API de autorização.

O serviço que fornece os dados do *usuário* (*perfis* e *transações*) é o serviço **Recupera as informações do usuário para o processo de autorização**. Ele retorna os dados em um token JWT assinado.

Não é mandatório que o sistema valide a assinatura do token. Você pode simplesmente decodificar o payload do token em Base64 e recuperar os dados. Mas é recomendável validar, para garantir o não repúdio da informação. O serviço que fornece a chave pública a ser utilizada na validação da assinatura é o **Recupera a chave pública do Autoriza utilizada para assinar os tokens de usuário e de sistema**.

O acesso ao serviço é validado através da passagem do token de acesso recuperado do **Acesso gov.br** no *header Authorization* da requisição.

> **Recupera as informações do usuário para o processo de autorização**
>
> Como exemplo, listamos uma chamada ao serviço **Recupera as informações do usuário para o processo de autorização**, com o *client id* do sistema igual a **autorizagov.estaleiro.serpro.gov.br** e o *id do usuário* igual a **27458032049**:
>
> ```
> curl -X GET "https://h.api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/sistemas/autorizagov.estaleiro.serpro.gov.br/
> usuarios/27458032049/infoUsuario" -H "accept: application/json" -H  "Authorization: Bearer
> eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGVu
> I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
> zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
> zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ"
> ```
>
> Ao executar o exemplo, substitua o *client id*, o *id do usuário* e o *token* no *header Authorization* pelo *client id* da sua aplicação, o seu *CPF* e o *token de acesso* recuperado do **Acesso gov.br**, respectivamente.

Execução de chamadas da API online

Para realização de testes de acessos mais simples, a API pode ser chamada diretamente através da página de referência da api, serviços de *autorizacao*.

1. Escolher o ambiente de execução no combo **Servers**
2. Clicar no botão **Authorize** e, no **BearerAuth**, colar a string do token de acesso do **Acesso gov.br** para o sistema e usuário. Clicar em **Authorize** e **Close**.
3. Escolher um serviço, clicar em **Try it Out**, preencher os parâmetros do serviço e clicar em **Execute**.

> **Cuidado ao executar a chamada online**
>
> Lembre-se que ao escolher um servidor e executar um teste, qualquer dado solicitado será efetivamente recuperado na respectiva base de dados. É sua responsabilidade usar o recurso online criteriosamente.

3.3 Integração com o Provider

3.3.1 Provider

O **Autoriza** pode atuar como um provedor de funções para **autorização** de acesso no lado cliente.

O controle da **autorização** fica de fato com o **Autoriza**, a aplicação parceira apenas demanda a aplicação das políticas de acesso, que é realizada no próprio contêiner de execução da aplicação.

Os providers são bibliotecas a serem empacotadas com a aplicação parceira. Eles oferecem uma interface programática para verificação do controle de acesso, com a disponibilização de funções (métodos) para recuperação dos dados de um usuário e aplicação das políticas de acesso durante o tratamento de requisições por uma aplicação. Um sistema, ao se integrar ao **Autoriza**, deve acionar essas funções, parametrizando-as com os dados do usuário e do sistema, para que o controle de acesso seja realizado.

Nesse cenário, o sistema utilizará a biblioteca provider específica para a linguagem e tecnologia da aplicação parceira, conforme demonstrado no exemplo de uso do provider java.

Os providers disponibilizam de forma programática métodos equivalentes aos serviços oferecidos pela API de validação. Eles fazem a aplicação das políticas de acesso, de modo equivalente à API. Mas, diferente da API, nesse caso a execução é feita no contêiner da aplicação cliente.

Conforme falado no início, o *sistema* precisa resolver a autenticação do *usuário* antes de acionar o **Autoriza**. Ou seja, se o *usuário* ainda não fez o login, ele precisa fazer.

O fluxo de troca de mensagens de controle de acesso para um usuário **ainda não logado** na aplicação está representado no diagrama a seguir.

Autoriza como fornecedor dos dados de autorização e da engine para aplicação das políticas de acesso para um usuário ainda não autenticado

Caso o *usuário* já esteja logado na aplicação, ela já pode acionar diretamente as funções do provider **Autoriza**.

O fluxo de troca de mensagens de controle de acesso para um usuário **já logado** na aplicação está representado no diagrama a seguir.

Autoriza como fornecedor dos dados de autorização e da engine para aplicação das políticas de acesso para um usuário já autenticado

> **Auditoria**
>
> Ao adotar esse modelo, a aplicação parceira fica responsável por implementar a auditoria dos acessos realizados por um usuário.
>
> Criando um logger "AUTORIZA" no contêiner da aplicação, é possível habilitar o nível "DEBUG" e rastrear as avaliações de acesso feitas pelo Autoriza. Não recomendamos que esses dados sejam usados como auditoria. Pensamos neles como facilitadores para análises de problemas de acesso de um usuário.

Providers disponíveis

Linguagem	Versão	Respositório onde está disponível	Tecnologias específicas
Java	8	http://nexus.aic.serpro/#browse/browse:autoriza-nexus	→ Java puro → Servlet filter → Anotações com AspectJ → Anotações com JavaEE

O que é preciso cadastrar no Autoriza?

Para possibilitar essa integração, é preciso que seja realizado, para o *sistema*, o cadastro das seguintes entidades no **Autoriza**:

- Sistema
- Perfis
- Políticas de acesso
- Transações (opcional)
- Atribuições de perfis a usuários
- Features e características do usuário (opcional)

Benefícios

Possibilita ao gestor a administração centralizada da criação de perfis e transações, atribuição desses aos usuários, e definição das políticas de acesso

Possibilita ao gestor o rastreamento das mudanças realizadas nos cadastros de perfis, transações, políticas de acesso e atribuições, através da auditoria

Possibilita ao gestor um completa governança da **autorização**. Mudanças nas políticas de acesso da aplicação não precisam de um novo release

Delega para o **Autoriza** a manutenção, evolução e suporte dos cadastros relacionados à **autorização**, bem como da lógica necessária para aplicação das políticas de acesso, possibilitando à aplicação parceira um maior foco nas demandas diretamente relacionadas aos seus fluxos de negócio

Limitações

Modelo de integração mais invasivo, uma vez que é preciso empacotar uma biblioteca do **Autoriza** com o backend da sua aplicação

A engine de aplicação de políticas é executada do lado cliente, consumindo os recursos de infra da aplicação

Depende da disponibilidade da biblioteca específica para a linguagem e tecnologia empregadas no sistema

Cenários indicados para uso

O uso desse modelo de integração é o mais indicado para a maioria dos cenários. Só não seria recomendado, a princípio, por alguma limitação de oferta ou incompatibilidade tecnológica de uso da biblioteca no contêiner da aplicação (incompatibilidade das versões de java, por exemplo).

Uso do provider

O uso do provider java, que necessita de configurações no ambiente de execução, será tratado em seção específica.

3.3.2 Uso do provider Java

Provider Java - Interface

O provider java é composto por uma série de projetos:

- autoriza-java-provider-basico
- autoriza-java-provider-anotacoes
- autoriza-java-provider-filtro

A utilização do provider básico é obrigatória.

As bibliotecas para filtro e para anotações são de uso opcional. Elas estendem o provider básico, disponibilizando um *servlet filter* e uma anotação java, respectivamente.

AUTORIZA-JAVA-PROVIDER-BASICO

É o projeto básico que disponibiliza de forma programática métodos equivalentes aos serviços oferecidos pela API de validação.

Ele segue a arquitetura de referência XACML.

Para utilizá-lo, é preciso que sejam definidos os valores das configurações, conforme seção específica.

Utilização

Importação no *pom.xml*:

```
<dependency>
  <groupId>br.gov.serpro.autoriza</groupId>
  <artifactId>autoriza-java-provider-basico</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
```

Interface disponível

Todos os métodos disponíveis recebem o CPF do usuário e o token de acesso do usuário recuperado do **Acesso gov.br**.

Eles não possuem retorno, e executam com sucesso caso o usuário **possua acesso** ao recurso solicitado, ou retornam a exceção *ExcecaoAcessoNaoAutorizado* caso o usuário **não possua acesso**.

Os métodos estão disponíveis na instância da classe *PontoExecucaoPoliticas*, conforme exemplo de uso.

A documentação dos métodos segue listada a seguir:

```
  /**
   * Aplica as políticas de acesso do sistema para verificar se o usuário pode
   * acessar o recurso solicitado com o verbo http.
   * O método lança uma ExcecaoAcessoNaoAutorizado caso o usuário não possua
   * acesso ao recurso soliticitado.
   *
   * @param idUsuario CPF do usuário logado na aplicação. Deve ser o mesmo CPF
   * presente no token de acesso do Acesso gov.br
   * @param caminhoRecurso Caminho do recurso na aplicação. Esse caminho deve
   * serguir o mesmo padrão do que está definido no arquivo de políticas.
   * Consulte o arquivo de políticas da aplicação para saber se está passando
   * o caminho de modo coerente com ele.
   * @param verboHTTP Verbo HTTP utilizado na requisição do usuário.
   * @param token Token de acesso JWT do usuário recuperado do Acesso gov.br.
   * @param filtrosOpcionais Mapa com os valores de filtro a serem considerados
   * na filtragem dos perfis atribuídos aos usuários, conforme as features
   * definidas para o sistema
   *
   * @exception ExcecaoAcessoNaoAutorizado Lançada quando o usuário não possui
   * acesso ao recurso solicitado.
   * @exception ExcecaoAutoriza Lançada quando ocorre algum erro durante a
   * avaliação das políticas de acesso.
   */
  void avaliarPoliticas(String idUsuario, String caminhoRecurso,
    String verboHTTP, String token, Map<String, Object> filtrosOpcionais);

  /**
   * Verifica se o usuário possui alguma atribuição com um determinado perfil.
   * O método lança uma ExcecaoAcessoNaoAutorizado caso o usuário não possua o
   * perfil soliticitado.
   *
```

```
     * @param idUsuario CPF do usuário logado na aplicação. Deve ser o mesmo CPF
     * presente no token de acesso do Acesso gov.br
     * @param nmPerfil Nome do perfil a ser verificado.
     * @param token Token de acesso JWT do usuário recuperado do Acesso gov.br.
     * @param filtrosOpcionais Mapa com os valores de filtro a serem considerados
     * na filtragem dos perfis atribuídos aos usuários, conforme as features
     * definidas para o sistema.
     *
     * @exception ExcecaoAcessoNaoAutorizado Lançada quando o usuário não possui
     * o perfil soliticitado.
     * @exception ExcecaoAutoriza Lançada quando ocorre algum erro durante a
     * verificação.
     */
    void usuarioPossuiPerfil(String idUsuario, String nmPerfil, String token,
      Map<String, Object> filtrosOpcionais);

    /**
     * Verifica se o usuário possui alguma atribuição com uma determinada
     * transação.
     * O método lança uma ExcecaoAcessoNaoAutorizado caso o usuário não possua
     * a transação soliticitada.
     *
     * @param idUsuario CPF do usuário logado na aplicação. Deve ser o mesmo CPF
     * presente no token de acesso do Acesso gov.br
     * @param nmTransacao Nome da transação a ser verificada.
     * @param token Token de acesso JWT do usuário recuperado do Acesso gov.br.
     * @param filtrosOpcionais Mapa com os valores de filtro a serem considerados
     * na filtragem dos perfis atribuídos aos usuários, conforme as features
     * definidas para o sistema.
     *
     * @exception ExcecaoAcessoNaoAutorizado Lançada quando o usuário não possui
     * a transação soliticitada.
     * @exception ExcecaoAutoriza Lançada quando ocorre algum erro durante a
     * verificação.
     */
    void usuarioPossuiTransacao(String idUsuario, String nmTransacao,
      String token, Map<String, Object> filtrosOpcionais);

    /**
     * Verifica se o usuário possui alguma atribuição com ao menos um dos perfis
     * solicitados.
     * O método lança uma ExcecaoAcessoNaoAutorizado caso o usuário não possua ao
     * menos um entre os perfis soliticitados.
     *
     * @param idUsuario CPF do usuário logado na aplicação. Deve ser o mesmo CPF
     * presente no token de acesso do Acesso gov.br
     * @param nmsPerfis Lista com os nomes dos perfis a serem verificados.
     * @param token Token de acesso JWT do usuário recuperado do Acesso gov.br.
     * @param filtrosOpcionais Mapa com os valores de filtro a serem considerados
     * na filtragem dos perfis atribuídos aos usuários, conforme as features
     * definidas para o sistema.
     *
     * @exception ExcecaoAcessoNaoAutorizado Lançada quando o usuário não possui
     * ao menos um entre os perfis soliticitados.
     * @exception ExcecaoAutoriza Lançada quando ocorre algum erro durante a
     * verificação.
     */
    void usuarioPossuiAoMenosUmDosPerfis(String idUsuario, List<String> nmsPerfis,
      String token, Map<String, Object> filtrosOpcionais);

    /**
     * Verifica se o usuário possui alguma atribuição com uma ao menos uma das
     * transações solicitadas.
     * O método lança uma ExcecaoAcessoNaoAutorizado caso o usuário não possua
     * ao menos uma entre as transações soliticitadas.
     *
     * @param idUsuario CPF do usuário logado na aplicação. Deve ser o mesmo CPF
     * presente no token de acesso do Acesso gov.br
     * @param nmsTransacoes Nomes da transações a serem verificadas.
     * @param token Token de acesso JWT do usuário recuperado do Acesso gov.br.
     * @param filtrosOpcionais Mapa com os valores de filtro a serem considerados
     * na filtragem dos perfis atribuídos aos usuários, conforme as features
     * definidas para o sistema.
     *
     * @exception ExcecaoAcessoNaoAutorizado Lançada quando o usuário não possui
     * ao menos uma entre as transações soliticitadas.
     * @exception ExcecaoAutoriza Lançada quando ocorre algum erro durante a
     * verificação.
     */
    void usuarioPossuiAoMenosUmaDasTransacoes(String idUsuario,
      List<String> nmsTransacoes, String token, Map<String,
      Object> filtrosOpcionais);
```

Provider Java - Configurações

O projeto precisa de uma série de configurações que devem ser realizadas definindo as seguintes variáveis de ambiente:

CLASSE_CONFIGURACAO_INJECAO_DEPENDENCIAS → Classe que faz a configuração das injeções da aplicação, permitindo a extensão da solução conforme as necessidades da aplicação cliente. A classe deve estender a classe *com.google.inject.AbstractModule* da biblioteca *Google Guice*, e definir, no método *configure*, classes que implementem as seis interfaces padrão do provider, conforme o exemplo:

```
@Override
protected void configure() {
  bind(CacheAutoriza.class).to(CacheAutorizaPadrao.class);
  bind(FachadaAutoriza.class).to(FachadaAutorizaRest.class);
  bind(PontoAdministracaoPoliticas.class).to(PontoAdministracaoPoliticasServico.class);
  bind(PontoInformacaoPoliticas.class).to(PontoInformacaoPoliticasPadrao.class);
  bind(PontoDecisaoPoliticas.class).to(PontoDecisaoPoliticasPadrao.class);
  bind(PontoExecucaoPoliticas.class).to(PontoExecucaoPoliticasPadrao.class);
}
```

Valor padrão: *br.gov.serpro.autoriza.plataforma.configuracao.ConfiguracaoInjecaoPadrao*

AUTORIZA_SISTEMA_CLIENT_ID → Client id da aplicação no **Acesso gov.br**.
Valor que deve ser utilizado em ambientes não produtivos: *client id da aplicação no ambiente de staging do* **Acesso gov.br**.
Valor que deve ser utilizado em ambientes produtivos: *client id da aplicação no ambiente de produção do* **Acesso gov.br**.

AUTORIZA_SISTEMA_PADRAO_POLITICAS → Padrão de politicas a ser utilizado pela aplicação. Esse valor do arquivo de políticas só será utilizado se a configuração *default* for sobrescrita e a classe *PontoAdministracaoPoliticasArquivo* for utilizada na configuração de injeção de dependências.
Valor padrão: *perfil.nome,transacao.nome,recurso.caminho,http.verbo*

AUTORIZA_SISTEMA_CAMINHO_ARQUIVO_POLITICAS → Caminho do arquivo de políticas no path de execução. Esse valor do arquivo de políticas só será utilizado se a configuração *default* for sobrescrita e a classe *PontoAdministracaoPoliticasArquivo* for utilizada na configuração de injeção de dependências.
Valor padrão: */opt/arquivosconfig/autoriza-politicas.csv*

AUTORIZA_URL_CONSULTA_SISTEMA → URL da consulta de sistema no Autoriza.
Valor que deve ser utilizado em ambientes não produtivos: *https://h.api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/sistemas/{clientId}/infoSistema*
Valor que deve ser utilizado em ambientes produtivos: *https://api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/sistemas/{clientId}/infoSistema*

AUTORIZA_URL_CONSULTA_USUARIO → URL da consulta de usuário no Autoriza.
Valor que deve ser utilizado em ambientes não produtivos: *https://h.api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/sistemas/{clientId}/usuarios/{idUsuario}/infoUsuario*
Valor que deve ser utilizado em ambientes produtivos: *https://api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/sistemas/{clientId}/usuarios/{idUsuario}/infoUsuario*

AUTORIZA_URL_CONSULTA_CHAVE_PUBLICA → URL da consulta de chave pública no Autoriza.
Valor que deve ser utilizado em ambientes não produtivos: *https://h.api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/chavePublica*
Valor que deve ser utilizado em ambientes produtivos: *https://api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/autorizacao/chavePublica*

AUTORIZA_CACHE_SISTEMA_EXPIRACAO_SEGUNDOS → Tempo da expiração da cache de sistema.

O tempo de expiração dessa cache deve ser avaliado conforme os riscos de negócio associados à possibilidade de se trabalhar com políticas de acesso defasadas em relação àquelas cadastradas na interface administrativa do **Autoriza**.
Valor padrão: *600*

AUTORIZA_CACHE_USUARIOS_TAMANHO_UNIDADES → Quantidade de usuários na cache de usuários.
Esse tamanho de cache deve ser definido conforme a característica de demanda da aplicação.

> **Consumo de memória**
>
> A implementação padrão de cache do provider é feita em memória, com a utilização da biblioteca *Google Guava*, então é necessário ter ciência que o aumento do número de registros na cache vai implicar num aumento no consumo de memória pela aplicação.

Valor padrão: *1000*

AUTORIZA_CACHE_USUARIOS_EXPIRACAO_SEGUNDOS → Tempo da expiração da cache de usuários.
O tempo de expiração dessa cache deve ser avaliado conforme os riscos de negócio associados à possibilidade de se trabalhar perfis e transações de usuário defasados em relação àqueles cadastrados para o usuário na interface administrativa do **Autoriza**.
Valor padrão: *600*

AUTORIZA_CACHE_CHAVE_PUBLICA_EXPIRACAO_SEGUNDOS → Tempo da expiração da cache de chave pública.
Valor padrão: *3600*

Um arquivo de nome *autoriza-conf.properties* com a definição *default* dessas propriedade está presente no *jar* do provider java básico. A aplicação parceira pode criar um arquivo similar de mesmo nome no seu caminho raiz de classpath, ou utilizar variáveis de ambiente conforme já sugerimos.

No caso de haver a definição de uma variável de ambiente com o mesmo nome de uma propriedade no arquivo de configuração, a *engine* do **Autoriza** utilizará o valor da variável de ambiente.

Provider Java - Exemplo de Uso

Para chamar os métodos disponíveis, é preciso realizar a configuração e ter conhecimento da interface.

Dados de autenticação a serem guardados

Súa aplicação deve guardar, para utilizar nas chamadas ao provider do Autoriza:
- ✓ o CPF do usuário logado
- ✓ o token de acesso do **Acesso gov.br**

As chamadas a cada um dos métodos para validação de acesso são exemplificadas a seguir.

Avaliação de políticas

Para aplicar as políticas de acesso a um recurso da aplicação, o método a ser executado é o avaliarPoliticas.

O exemplo a seguir possui parâmetros fixos, apenas a título de exemplo. No uso na sua aplicação, recupere os parâmetros da requisição que está sendo processada no momento.

```
String cpfUsuario = "27458032049";
String caminhoRecurso = "/private/protocolo";
String verboHTTP = "GET";
String tokenAcessoGov =
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGV
I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ";
Map<String, Object> filtrosOpcionais = null;

PontoExecucaoPoliticas pep = FabricaInjecao.getInstancia().getBean(PontoExecucaoPoliticas.class);
pep.avaliarPoliticas(cpfUsuario, caminhoRecurso, verboHTTP, tokenAcessoGov, filtrosOpcionais);
```

Verificar se o usuário possui um determinado perfil

Caso você prefira definir suas políticas de acesso no código da sua aplicação, você pode perguntar ao **Autoriza** diretamente se o usuário possui um determinado perfil.

O exemplo a seguir possui parâmetros fixos, apenas a título de exemplo. No uso na sua aplicação, defina o perfil a ser consultado conforme a sua necessidade.

```
String cpfUsuario = "27458032049";
String nmPerfil = "GESTOR PROTOCOLO";
String tokenAcessoGov =
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGV
I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ";
Map<String, Object> filtrosOpcionais = new HashMap<String, Object>();
filtrosOpcionais.put("cdOrgaoSiape", 17000l);

PontoExecucaoPoliticas pep = FabricaInjecao.getInstancia().getBean(PontoExecucaoPoliticas.class);
pep.usuarioPossuiPerfil(cpfUsuario, nmPerfil, tokenAcessoGov, filtrosOpcionais);
```

No exemplo acima, só serão considerados os perfis atribuídos cujo valor da **Órgão SIAPE** é *17000*.

Verificar se o usuário possui acesso a uma determinada transação

Caso você prefira definir suas políticas de acesso no código da sua aplicação, você também pode perguntar ao **Autoriza** diretamente se o usuário possui uma determinada transação.

O exemplo a seguir possui parâmetros fixos, apenas a título de exemplo. No uso na sua aplicação, defina a transação a ser consultada conforme a sua necessidade.

```
String cpfUsuario = "27458032049";
String nmTransacao = "MANTER PROTOCOLO";
String tokenAcessoGov =
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGV
I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ";
Map<String, Object> filtrosOpcionais = new HashMap<String, Object>();
filtrosOpcionais.put("protocolo", "1234567890");

PontoExecucaoPoliticas pep = FabricaInjecao.getInstancia().getBean(PontoExecucaoPoliticas.class);
pep.usuarioPossuiTransacao(cpfUsuario, nmTransacao, tokenAcessoGov, filtrosOpcionais);
```

No exemplo acima, só serão considerados os perfis atribuídos cujo valor da feature configurável **protocolo** é "*1234567890*".

Verificar se o usuário possui ao menos um dos perfis dentro de uma lista de perfis

Caso você prefira definir suas políticas de acesso no código da sua aplicação, você também pode perguntar ao **Autoriza** diretamente se o usuário possui ao menos um entre uma lista de perfis.

O exemplo a seguir possui parâmetros fixos, apenas a título de exemplo. No uso na sua aplicação, defina os perfis a serem consultados conforme a sua necessidade.

```
String cpfUsuario = "27458032049";
String nmsPerfis = Arrays.asList(new String[]{"GESTOR PROTOCOLO", "ADMINISTRADOR"});
String tokenAcessoGov =
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGV
I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ";
Map<String, Object> filtrosOpcionais = new HashMap<String, Object>();
filtrosOpcionais.put("cdOrgaoSiape", 17000l);
filtrosOpcionais.put("protocolo", "1234567890");
filtrosOpcionais.put("tag", "GESTAO");
filtrosOpcionais.put("idAtribuicao", 789123l);

PontoExecucaoPoliticas pep = FabricaInjecao.getInstancia().getBean(PontoExecucaoPoliticas.class);
pep.usuarioPossuiAoMenosUmDosPerfis(cpfUsuario, nmsPerfis, tokenAcessoGov, filtrosOpcionais);
```

No exemplo acima, só serão considerados os perfis atribuídos cujo valor da feature configurável **protocolo** é "*1234567890*", e cujo valor do **Órgão SIAPE** é *17000*. Adicionalmente, só serão considerados os perfis que estejam vinculados à tag "*GESTAO*" e a atribuição de perfis (feature **Habilitação**) cujo identificador é *789123*.

Verificar se o usuário possui acesso a ao menos uma transação dentro de uma lista de transações

Caso você prefira definir suas políticas de acesso no código da sua aplicação, você também pode perguntar ao **Autoriza** diretamente se o usuário possui ao menos uma entre uma lista de transações.

O exemplo a seguir possui parâmetros fixos, apenas a título de exemplo. No uso na sua aplicação, defina as transações a serem consultadas conforme a sua necessidade.

```
String cpfUsuario = "27458032049";
String nmsTransacoes = Arrays.asList(new String[]{"MANTER PROTOCOLO", "CONSULTAR PROCESSOS"});
String tokenAcessoGov =
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGV
I6BSXduiDnE-OChTSDNOs4mQ23BuSfI5WbkJP1qGi7LoCTZq2barkONFqX5rtrqCvaFi_LbvwD0MygO1MZn_f6mbIhPa2tU_KJWGihmoDZEnzLZI-ualRBP0yym-
zsisnQrozqk3eVFrPJ4dvzcG5Te9GkgZV0KmIwYyBxJTu5W1EYgg23w2UBElgZlAggSSY2DJIZ2ID_qSyZboyydT4y7L-
zrGE0RQabxm9saUQXnwsT9Hw49h9pZjakoo33SJIaWLxd6HtujC_m3dBKptTRr9XSv2rebj9wQQ";
Map<String, Object> filtrosOpcionais = new HashMap<String, Object>();
filtrosOpcionais.put("cdOrgaoSiape", 17000l);
filtrosOpcionais.put("protocolo", "1234567890");
filtrosOpcionais.put("cnpj", "54168882000128");
filtrosOpcionais.put("tag", "CONTABILIDADE");

PontoExecucaoPoliticas pep = FabricaInjecao.getInstancia().getBean(PontoExecucaoPoliticas.class);
pep.usuarioPossuiAoMenosUmaDasTransacoes(cpfUsuario, nmsTransacoes, tokenAcessoGov, filtrosOpcionais);
```

No exemplo acima, só serão considerados os perfis atribuídos cujo valor da da feature configurável **protocolo** é "*1234567890*", cujo valor do **Órgão SIAPE** é *17000*, e cujo valor da feature configurável **cnpj** é "*54168882000128*". Adicionalmente, só serão considerados os perfis que estejam vinculados à tag "*CONTABILIDADE*".

Provider Java - Filtros

O projeto **autoriza-java-provider-filtro** possui um *servlet filter* para interceptar métodos e verificar se o usuário possui o perfil ou transação configurados, nas políticas de acesso ao sistema, para acesso ao recurso.

Com o filtro, é possível utilizar as políticas de acesso da aplicação cadastradas no **Autoriza** sem a necessidade da chamada direta aos métodos do provider java básico.

CONFIGURAÇÃO

O projeto de filtro não precisa de variáveis de ambiente além daquelas já definidas para o provider java básico.

UTILIZAÇÃO

Importação no *pom.xml*:

```
<dependency>
  <groupId>br.gov.serpro.autoriza</groupId>
  <artifactId>autoriza-java-provider-filtro</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
```

Para usar o filtro, é necessário declará-lo no *web.xml* da sua aplicação conforme o exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">
   <display-name>Servlet Application Example</display-name>
   <filter>
      <filter-name>FiltroAutoriza</filter-name>
      <filter-class>br.gov.serpro.autoriza.filtro.FiltroAutoriza</filter-class>
   </filter>
   <filter-mapping>
      <filter-name>FiltroAutoriza</filter-name>
      <url-pattern>/private/*</url-pattern>
   </filter-mapping>
</web-app>
```

Atentar para o *pattern* que deseja utilizar, para que o filtro fique restrito a determinado *path* de conteúdo da sua aplicação.

Adicionalmente, é preciso definir uma classe que implemente a interface *DadosUsuarioFiltroResolver*, e defina os métodos:

```
  /**
   * Retorna o id do usuário logado (CPF)
   *
   * @param servletRequest Dados do request do usuário
   *
   * @return CPF do usuário logado
   */
  String getIdUsuarioLogado(HttpServletRequest servletRequest);

  /**
   * Retorna o token de acesso do Acesso gov.br para o usuário
   *
   * @param servletRequest Dados do request do usuário
   *
   * @return Token de acesso do acesso gov para o usuário
   */
  String getTokenAcessoGov(HttpServletRequest servletRequest);
```

Essa realização da interface deve ser utilizada na classe de configuração de injeção de dependências do provider, de um modo similar a:

```
  @Override
  protected void configure() {
    bind(DadosUsuarioFiltroResolver.class).to(DadosUsuarioFiltroResolverImpl.class);
    bind(CacheAutoriza.class).to(CacheAutorizaPadrao.class);
    bind(FachadaAutoriza.class).to(FachadaAutorizaRest.class);
    bind(PontoAdministracaoPoliticas.class).to(PontoAdministracaoPoliticasServico.class);
    bind(PontoInformacaoPoliticas.class).to(PontoInformacaoPoliticasPadrao.class);
    bind(PontoDecisaoPoliticas.class).to(PontoDecisaoPoliticasPadrao.class);
    bind(PontoExecucaoPoliticas.class).to(PontoExecucaoPoliticasPadrao.class);
  }
```

O filtro retorna uma exceção *ExcecaoAcessoNaoAutorizado* no caso do usuário não possuir acesso ao recurso.

Provider Java - Anotações

Os projetos **autoriza-java-provider-anotacoes** e **autoriza-java-provider-anotacoes-javaee** possuem anotações para interceptar métodos e verificar se o usuário possui determinado perfil ou transação. O primeiro utiliza AspectJ, e é o mais indicado para utilizar com *frameworks* que utilizam AspectJ, como o Spring Framework, por exemplo. O segundo utiliza anotações JavaEE, e é indicado para projetos que utilizam essa tecnologia.

Com as anotações, as políticas de acesso definidas no **Autoriza** não são utilizadas. Ao invés disso, é o próprio código da aplicação cliente que define as políticas.

CONFIGURAÇÃO

O projeto de filtro não precisa de variáveis de ambiente além daquelas já definidas para o provider java básico.

UTILIZAÇÃO

Importação no *pom.xml*:

```
<dependency>
  <groupId>br.gov.serpro.autoriza</groupId>
  <artifactId>autoriza-java-provider-anotacoes</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
```

ou

```
<dependency>
  <groupId>br.gov.serpro.autoriza</groupId>
  <artifactId>autoriza-java-provider-anotacoes-javaee</artifactId>
  <version>1.0.0-SNAPSHOT</version>
</dependency>
```

Se você estiver utilizando JavaEE, é preciso declarar os interceptors no *beans.xml* da sua aplicação, de modo similar a:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
    <interceptors>
        <class>br.gov.serpro.autoriza.interceptadores.InterceptadorPerfis</class>
    </interceptors>
        <interceptors>
        <class>br.gov.serpro.autoriza.interceptadores.InterceptadorTransacoes</class>
    </interceptors>
</beans>
```

Adicionalmente, no caso do **autoriza-java-provider-anotacoes**, é preciso definir uma classe que implemente a interface *br.gov.serpro.autoriza.anotacao.resolver.DadosUsuarioAnotacaoResolver*, e defina os métodos:

```
  /**
   * Retorna o id do usuário logado (CPF)
   *
   * @param joinPoint Join point de execução da anotação
   *
   * @return CPF do usuário logado
   */
  String getIdUsuarioLogado(ProceedingJoinPoint joinPoint);

  /**
   * Retorna o token de acesso do Acesso GovBR para o usuário
   *
   * @param joinPoint Join point de execução da anotação
   *
   * @return Token de acesso do acesso gov para o usuário
   */
  String getTokenAcessoGov(ProceedingJoinPoint joinPoint);
```

No caso do **autoriza-java-provider-anotacoes-javaee**, é preciso definir uma classe que implemente a interface *br.gov.serpro.autoriza.anotacao.javaee.resolver.DadosUsuarioAnotacaoResolver*, e defina os métodos:

```
  /**
   * Retorna o id do usuário logado (CPF)
   *
   * @return CPF do usuário logado
   */
  String getIdUsuarioLogado();
```

```
/**
 * Retorna o token de acesso do Acesso GovBR para o usuário
 *
 *
 * @return Token de acesso do acesso gov para o usuário
 */
String getTokenAcessoGov();
```

Essa realização da interface deve ser utilizada na classe de configuração de injeção de dependências do provider, de um modo similar a:

```
@Override
protected void configure() {
  bind(DadosUsuarioAnotacaoResolver.class).to(DadosUsuarioAnotacaoResolverImpl.class);
  bind(CacheAutoriza.class).to(CacheAutorizaPadrao.class);
  bind(FachadaAutoriza.class).to(FachadaAutorizaRest.class);
  bind(PontoAdministracaoPoliticas.class).to(PontoAdministracaoPoliticasServico.class);
  bind(PontoInformacaoPoliticas.class).to(PontoInformacaoPoliticasPadrao.class);
  bind(PontoDecisaoPoliticas.class).to(PontoDecisaoPoliticasPadrao.class);
  bind(PontoExecucaoPoliticas.class).to(PontoExecucaoPoliticasPadrao.class);
}
```

Por fim, para utilizar os interceptadores no código Java, é só decorar o método desejado com as anotações *RequerPerfil* ou *RequerTransacao*, de modo equivalente a:

```
@RequerPerfil(perfis={"PERFIL_A","PERFIL_B"})
public void meuMetodoASerInterceptado() {

}

@RequerTransacao(transacoes={"TRANSACAO_B","TRANSACAO_C"})
public void meuMetodoASerInterceptadoDois() {

}
```

Os interceptadores retornam uma exceção *ExcecaoAcessoNaoAutorizado* no caso do usuário não possuir acesso ao perfil ou transação.

3.4 Aplicação Remota das Políticas de Acesso

3.4.1 Aplicação Remota das Políticas de Acesso

O **Autoriza** pode atuar como um provedor de serviços para **autorização** remota de acesso.

O controle da **autorização** fica de fato com o **Autoriza**, a aplicação parceira apenas demanda a aplicação das políticas de acesso.

Nesse cenário, o sistema utilizará os serviços da API de validação, conforme detalhado no exemplo de uso.

Conforme falado no início, o *sistema* precisa resolver a autenticação do *usuário* antes de acionar o **Autoriza**. Ou seja, se o *usuário* ainda não fez o login, ele precisa fazer.

O fluxo de troca de mensagens de controle de acesso para um usuário **ainda não logado** na aplicação está representado no diagrama a seguir.

Autoriza como serviço de aplicação remota das políticas de acesso para um usuário ainda não autenticado

Caso o *usuário* já esteja logado na aplicação, ela já pode acionar diretamente os serviços do **Autoriza**.

O fluxo de troca de mensagens de controle de acesso para um usuário **já logado** na aplicação está representado no diagrama a seguir.

Autoriza como serviço de aplicação remota das políticas de acesso para um usuário já autenticado

O que é preciso cadastrar no Autoriza?

Para possibilitar essa integração, é preciso que seja realizado, para o *sistema*, o cadastro das seguintes entidades no **Autoriza**:

- Sistema
- Perfis
- Políticas de acesso
- Transações (opcional)
- Atribuições de perfis a usuários
- Features e características do usuário (opcional)

Benefícios

Possibilita ao gestor a administração centralizada da criação de perfis e transações, atribuição desses aos usuários, e definição das políticas de acesso

Possibilita ao gestor o rastreamento das mudanças realizadas nos cadastros de perfis, transações, políticas de acesso e atribuições, bem como das autorizações de acesso realizadas, através da auditoria

Possibilita ao gestor um completa governança da **autorização**. Mudanças nas políticas de acesso da aplicação não precisam de um novo release

Delega para o **Autoriza** a manutenção, evolução e suporte dos cadastros relacionados à **autorização**, bem como da lógica necessária para aplicação das políticas de acesso, possibilitando à aplicação parceira um maior foco nas demandas diretamente relacionadas aos seus fluxos de negócio

Limitações

Maior granularidade de chamadas remotas entre os modelos de integração, onerando a performance da aplicação parceira

Menor flexibilidade em relação a adoção de lógicas específicas do negócio durante o controle de acesso

Cenários indicados para uso

O uso desse modelo de integração, por ser o de que mais demanda chamadas remotas ao **Autoriza**, é recomendado apenas para aplicações com uma única persona, ou para um controle mais restrito a alguma funcionalidade aplicação.

Por exemplo, sua aplicação só precisa validar o acesso do usuário uma vez, quando ele loga, para verificar se ele possui um determinado perfil de acesso. Após essa validação, o usuário terá acesso total à aplicação.

Outro exemplo, você tem uma aplicação com várias telas e componentes de tela, mas tem uma funcionalidade chamada "Limpar Base de Dados", a ser usada com bastante parcimônia, que é acessível apenas para usuários com uma determinada transação. Você decide adotar esse modelo de integração exclusivamente para a funcionalidade "Limpar Base de Dados".

> **Utilizar apenas em fluxos que possuem um número de validações de autorização bem restrito**
>
> A adoção desse modelo não é sustentável por uma típica aplicação de governo, com um bom número de personas, casos de uso e componentes de tela, uma vez que cada interação do usuário aciona uma chamada remota ao Autoriza, inviabilizando uma boa performance da aplicação.

3.4.2 Exemplo de Uso para Aplicação Remota das Políticas de Acesso

A API utilizada é a API de validação.

O principal serviço que implementa a engine de validação das políticas de acesso é o serviço **Avalia as políticas de acesso para o usuário**. Ele retorna um indicador boolean informando se o acesso foi autorizado ou não.

O acesso ao serviço é validado através da passagem do token de acesso recuperado do **Acesso gov.br** no *header Authorization* da requisição.

> **Avalia as políticas de acesso para o usuário**
>
> Como exemplo, listamos uma chamada ao serviço **Avalia as políticas de acesso para o usuário**, com o *client id* do sistema igual a **autorizagov.estaleiro.serpro.gov.br** e o *id do usuário* igual a **27458032049**, o *caminho* igual a **/private/protocolo** (em formato URI Encode) e *verbo* igual a **GET**:
>
> ```
> curl -X GET "https://h.api.autorizagov.estaleiro.serpro.gov.br/api-autoriza/validacao/sistemas/autorizagov.estaleiro.serpro.gov.br/usuarios/27458032049/
> caminho/%2Fprivate%2Fprotocolo/verbo/GET/avaliaPoliticas" -H "accept: application/json" -H  "Authorization: Bearer
> eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIyNzQ1ODAzMjA0OSIsImF1ZCI6ImF1dG9yaXphZ292LmVzdGFsZWlyby5zZXJwcm8uZ292LmJyIiwic2NvcGUiOlsiZW1haWwiLCJvcGVu
> XGW7MH4ny_JEa4YX6XNINg6kDXFHluEmnKVVsdld3I_UbeAHgfEw3p1mLU8GAzLMQRFHv3ntwtjCQt_9pYkrM0lfbrBQI-
> EVkNJpBP5KAS8QzBWMwJ2Nd_CZEU7KZV76rUDdANW9nC7XTDU2XmDq0xeBB1qr5D5cSm0WNvZFy3ruUKxNjDJ9-
> ODWAqd5y38tKMYE0LyCxyEKBIFyOwA28H4KVWh9Fx79QezyAfj0f6SEDX13I0y8W7gSODlEz-VVDbgA0GGytPVFM2oyQj3pvfhvoN1IUDgnSs4NV_zXPpp67PkMKpjU4nkwVVUg"
> ```
>
> Ao executar o exemplo, substitua o *client id*, o *id do usuário*, o *caminho do recurso*, o *verbo* e o *token* no *header Authorization* pelo *client id* da sua aplicação, o seu *CPF*, o *caminho* relativo do recurso na sua aplicação, o *verbo* HTTP utilizado na requisição e o *token de acesso* recuperado do **Acesso gov.br**, respectivamente.

Execução de chamadas da API online

Para realização de testes de acessos mais simples, a API pode ser chamada diretamente através da página de referência da api, serviços de *validacao*.

1. Escolher o ambiente de execução no combo **Servers**
2. Clicar no botão **Authorize** e, no **BearerAuth**, colar a string do token de acesso do **Acesso gov.br** para o sistema e usuário. Clicar em **Authorize** e **Close**.
3. Escolher um serviço, clicar em **Try it Out**, preencher os parâmetros do serviço e clicar em **Execute**.

> **Cuidado ao executar a chamada online**
>
> Lembre-se que ao escolher um servidor e executar um teste, qualquer dado consultado será efetivamente recuperado na respectiva base de dados, e serão gerados registros de auditoria para as chamadas. É sua responsabilidade usar o recurso online criteriosamente.

4. Adesão

Qualquer sistema de governo já integrado ao **Acesso gov.br** pode se tornar uma aplicação parceira do **Autoriza**.

4.1 Como se tornar uma aplicação parceira do Autoriza

Tornar-se um consumidor do **Autoriza** é simples, basta enviar um email para lista *lista-autorizagovbr@serpro.gov.br*, com o assunto **Adesão ao Autoriza**, informando os dados da aplicação (nome do sistema, código de serviço, responsável), incluindo o *client_id* da aplicação no **Acesso gov.br**.

Também é preciso informar no email:

- Modelo de adesão escolhido
- CPFs dos responsáveis
- Quantidade média de acessos diários esperada
- Período de pico de acesso
- Quantidade de acessos durante o período de pico

Esse email será respondido com os dados de cadastro da aplicação parceira nos ambientes de **Homologação** e **Produção** do **Autoriza**.

5. Boas práticas

Seguem listadas algumas boas práticas para uso do **Autoriza**.

5.1 Para os gestores

Os

gestores de sistema devem observar as seguintes orientações durante a manutenção dos dados da aplicação parceira.

Divisão do controle de acesso do sistema em perfis e transações


Os perfis devem corresponder às personas (ou atores) do seu sistema.

As transações devem corresponder às operações que os usuários podem realizar no seu sistema.

Granularidade das transações

✂
Seja pragmático. Dimensione a granularidade das transações conforme a real necessidade de segregação de acesso do seu sistema.

Por exemplo, se todos os usuários com acesso a um determinado caso de uso conseguem acessar todas as funcionalidades do caso de uso, é mais coerente ter uma única transação para esse caso de uso.

Dicas de nomenclatura


Seja sucinto ao nomear suas transações e perfis.

Deixe para entrar em maiores detalhes nas descrições.

Evite usar acentos ou símbolos nos nomes, para evitar problemas de codificação no uso das informações.

Sequência de cadastro sugerida

1. Cadastro das transações
2. Cadastro dos perfis e vinculação das transações
3. Definição das políticas de acesso
4. Configuração das features e características do usuário
5. Atribuição dos perfis aos usuários

5.2 Para os desenvolvedores

Os

desenvolvedores das *aplicações parceiras* devem observar as seguintes orientações no desenvolvimento da integração com o **Autoriza**.

Chamadas aos serviços

⚓
Seja responsável e nosso parceiro ao utilizar os serviços que oferecemos.

Caso o uso do seu sistema extrapole a previsão de acessos que inicialmente nos foi repassada, nos informe para que possamos revisar nossa infraestrutura tendo em vista garantir a sustentação das aplicações parceiras.

Expiração do token

📅
Controle o tempo de expiração do token de acesso do **Acesso gov.br** que serve como entrada para os serviços que oferecemos, de tal maneira a solicitar automaticamente outro quando o atual expirar.

URLs do Autoriza

🔠
Evite fixar no código as URLs dos *endpoints* do Autoriza.

Lembre que essas URLs não estão sob sua gestão e podem ser modificadas.

Quanto mais simples de alterá-las na sua aplicação, menor o impacto.

Verificação de conectividade

🔠 O Autoriza utiliza certificado Let's Encrypt, então verifique se a máquina a partir da qual você está tentando se conectar à API aceita a cadeia de certificação do Let's Encrypt.

Faça isso utilizando os exemplos de comando cURL listados nessa documentação.

Os certificados são renovados trimestralmente.

Consultoria com a equipe de sustentação

✉ Leia a documentação disponível, veja os exemplos de chamadas à API, faça testes locais.

A referência das APIs de autorização e validação deveria ser suficiente para sanar suas dúvidas em relação aos serviços.

Se ela não está sendo suficiente, nos informe, para que possamos melhorá-la. Faça isso enviando email para a lista *lista-autorizagovbr@serpro.gov.br*, com o assunto **Sugestão de melhoria para a integração com o Autoriza**.

Uso dos ambientes

🔠 Seus ambientes não produtivos devem apontar para o ambiente de Homologação do Autoriza.

O ambiente produtivo deve apontar para o ambiente de Produção.

Não aponte ambientes de teste ou homologação para Produção do Autoriza.

6. Referência

7. Release Notes

7.1 APIs de Autorização e Validação

v0.5.0 - 21/02/2020

- Versão inicial das APIs

v1.0.0 - 11/03/2020

- Correção de bugs
- Melhorias de sustentabilidade

v2.0.0 - 05/06/2020

- Correção de bugs
- Auditoria nas chamadas à API de validação

v2.1.0 - 09/12/2020

- Utilização de features

v2.2.0 - 16/03/2021

- Adição da feature Dias da Semana
- Adição da feature Vigência
- Adição da feature Horário

v2.3.0 - 20/05/2021

- Adição da feature Órgão SIAPE

v2.4.0 - 15/06/2021

- Adição da feature Tag
- Adição da feature Nivel de Acesso
- Adição da feature UORG SIAPE
- Adição da feature UPAG SIAPE

v2.5.0 - 30/06/2021

⚠
Versão disponível apenas no ambiente de homologação do **Autoriza**

- Adição da feature Habilitação

7.2 Provider Java

v0.5.0 - 21/02/2020

- Versão inicial do provider

v1.0.0 - 11/03/2020

- Correção de bugs
- Melhorias de sustentabilidade

v2.0.0 - 05/06/2020

- Correção de bugs
- Auditoria nas chamadas à API de validação

v2.1.0 - 09/12/2020

- Utilização de features

v2.2.0 - 16/03/2021

- Adição da feature Dias da Semana
- Adição da feature Vigência
- Adição da feature Horário

v2.3.0 - 20/05/2021

- Adição da feature Órgão SIAPE

v2.4.0 - 15/06/2021

- Adição da feature Tag
- Adição da feature Nivel de Acesso
- Adição da feature UORG SIAPE
- Adição da feature UPAG SIAPE

v2.5.0 - 30/06/2021

⚠
Versão disponível apenas no ambiente de homologação do **Autoriza**

- Adição da feature Habilitação