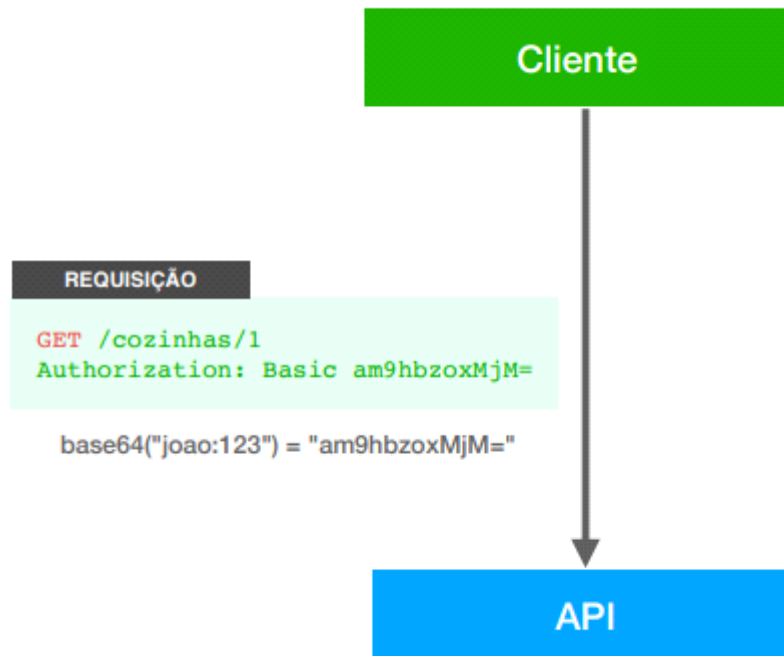


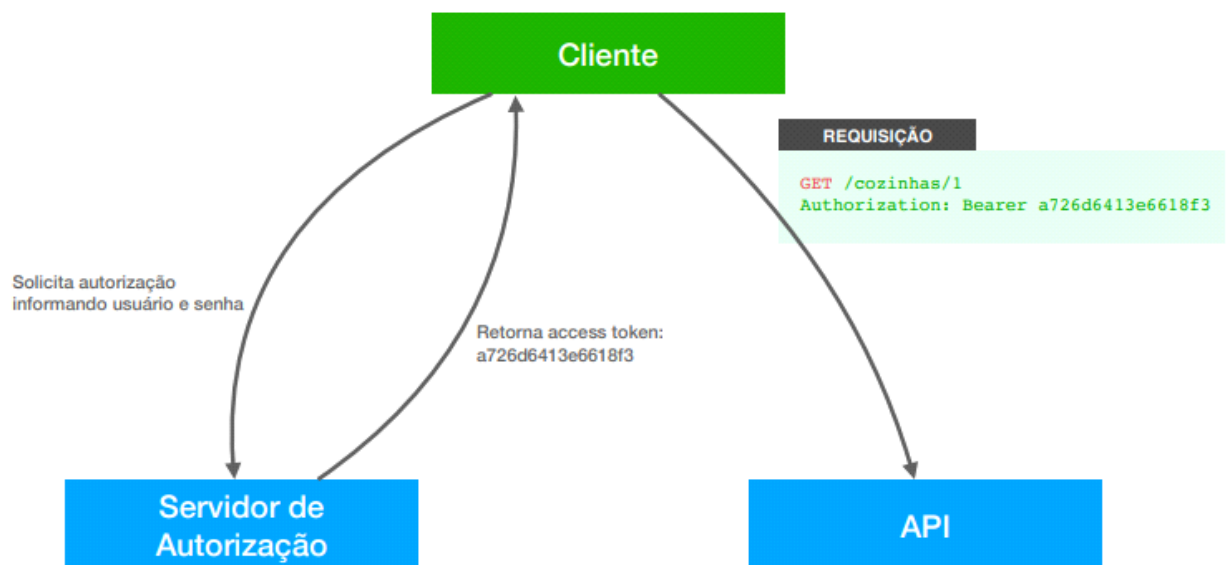
## 22.1. Introdução à segurança de REST APIs

sábado, 29 de abril de 2023 18:07

- HTTP Basic Authentication



- Oauth2



# 22.2. Adicionando segurança na API com Spring Security

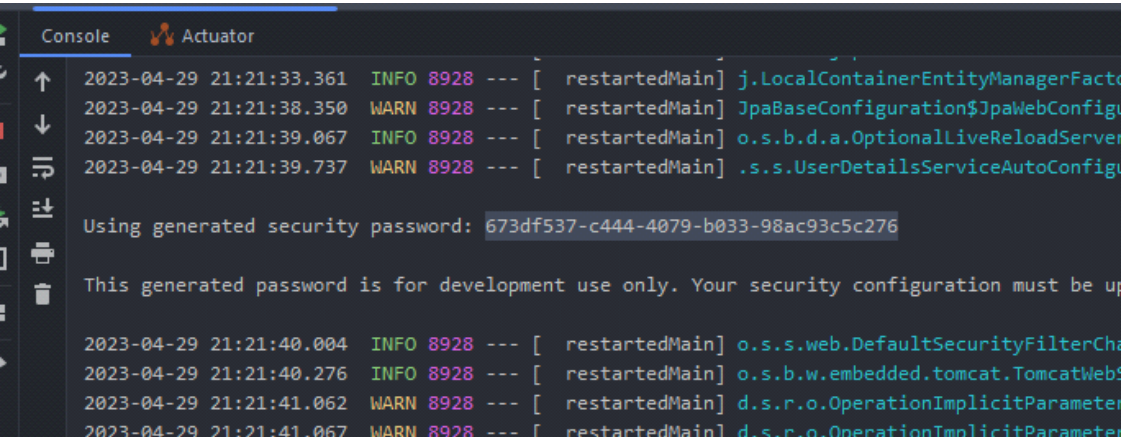
sábado, 29 de abril de 2023 18:12

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
  <version>3.0.6</version>
</dependency>
```

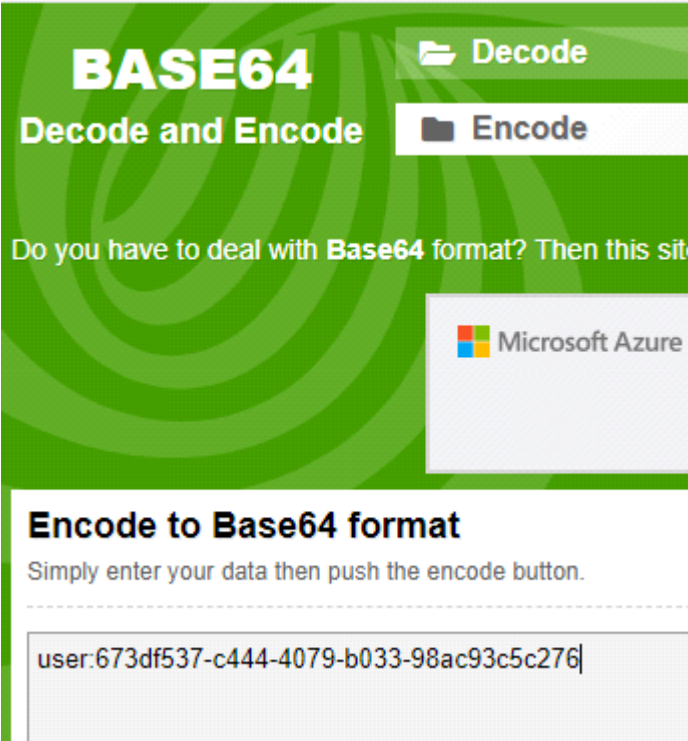
Configurando Http Basic Authentication

Ao adicionar a dependência do starter do Spring Security, automaticamente os endpoints são bloqueado e a resposta é um 401 Não autorizado

Ele fornece um user padrão e uma senha aleatória

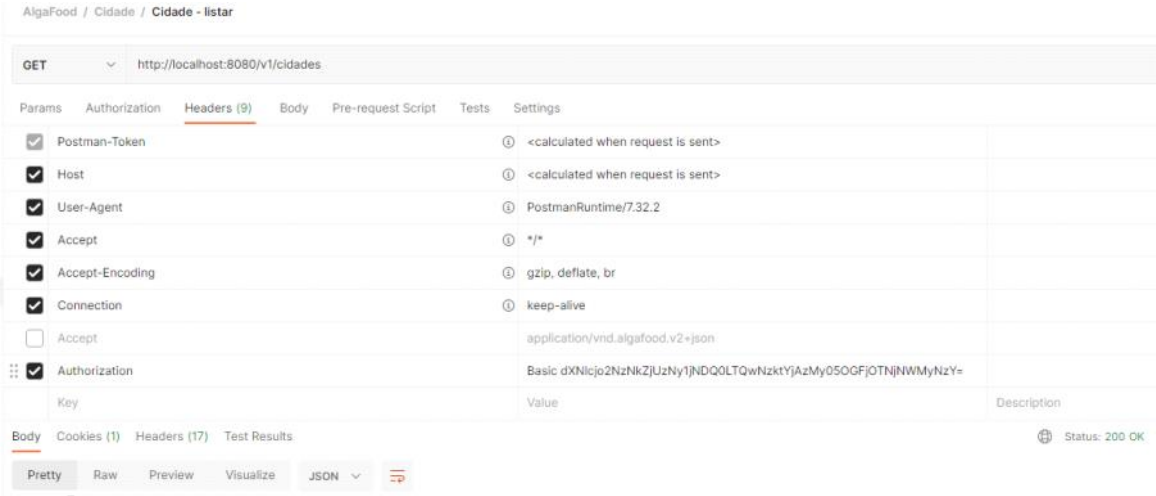


Como visto, ele aceita um usuario e senha encodados em base64



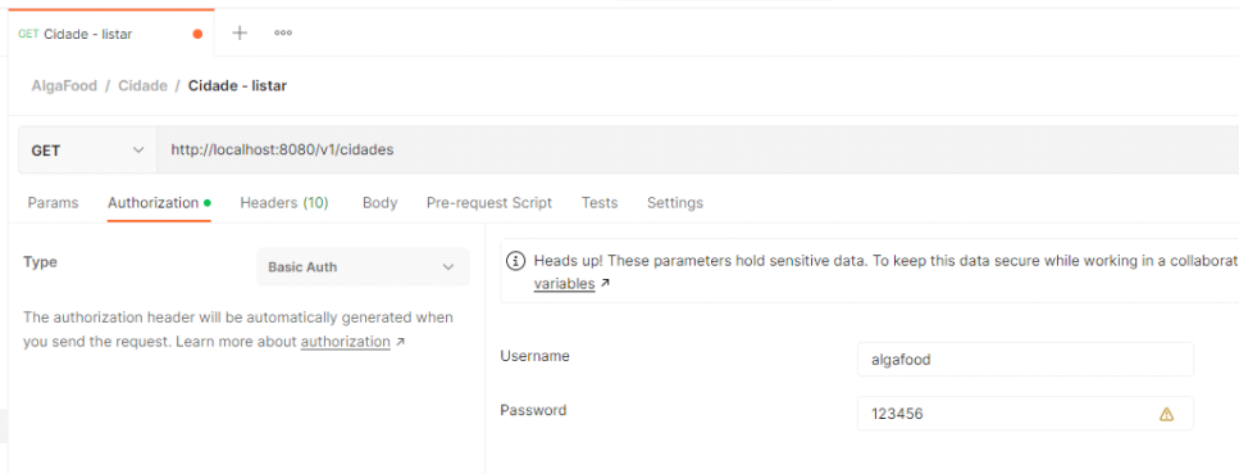
dXNlcjo2NzNkZjUzNy1jNDQ0LTQwNzktYjAzMy05OGFjOTNjNWMyNzY=

Temos que passar em um header Authorization o prefixo "Basic " seguido do token encodado em base64



Podemos alterar a senha e o password aleatório

```
#22.2. Adicionando segurança na API com Spring Security
spring.security.user.name=algafood
spring.security.user.password=123456
```



Temos um problema com cookies no postam ao utilizar o Basic Authorization, ele armazena sessões no postam e uma regra de uma Rest Api é o não armazenamento de sessões, uma aplicação statless, sem estado. Na próxima aula veremos como retirar essa configuração

# 22.4. Configurando autenticação de usuários em memória

sábado, 29 de abril de 2023 22:54

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /*Lista de usuários em memória para autenticação*/
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser( username: "Gustavo")
              .password("123")
              .roles("ADMIN")
            .and()
            .withUser( username: "Joao")
              .password("123")
              .roles("ADMIN");
    }
}
```

vamos substituir o "user" e a senha definida em application.properties

é necessário ter o método roles.

Criar Bean para encoding de senhas usando BCryptPasswordEncoder

```
/*Necessário para criptografar senha*/
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

```
/*Lista de usuários em memória para autenticação*/
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
        .withUser( username: "Gustavo")
          .password(passwordEncoder().encode( rawPassword: "123"))
          .roles("ADMIN")
        .and()
        .withUser( username: "Joao")
          .password(passwordEncoder().encode( rawPassword: "123"))
          .roles("ADMIN");
}
```

GET

http://localhost:8080/v1/cidades

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Type

Basic Auth

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collabora

variables

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Username

Gustavo

Password

123

## 22.3. Configurando Spring Security com HTTP Basic

sábado, 29 de abril de 2023 18:56

Configurações iniciais com HTTP Basic com Spring Security

```
package com.algaworks.algafood.core.security;

import ...

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.httpBasic()
            .and()
            .authorizeRequests()
            .anyRequest().authenticated();
    }
}
```

temos que sobrescrever o método configure com o parâmetro HttpSecurity.

estamos configurando o httpBasic e para todas as requisições, autorizar quando estiver autenticada.

Para permitir um endpoint sem autenticação

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.httpBasic()
            .and()
            .authorizeRequests()
            .antMatchers(...antPatterns: "/v1/cozinhas/**").permitAll()
            .anyRequest().authenticated();
    }
}
```

Para retirar o armazenamento de sessão

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.httpBasic()
        .and()
        .authorizeRequests()
        .antMatchers(...antPatterns: "/v1/cozinhas/**").permitAll()
        .anyRequest().authenticated()

        .and()
        .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

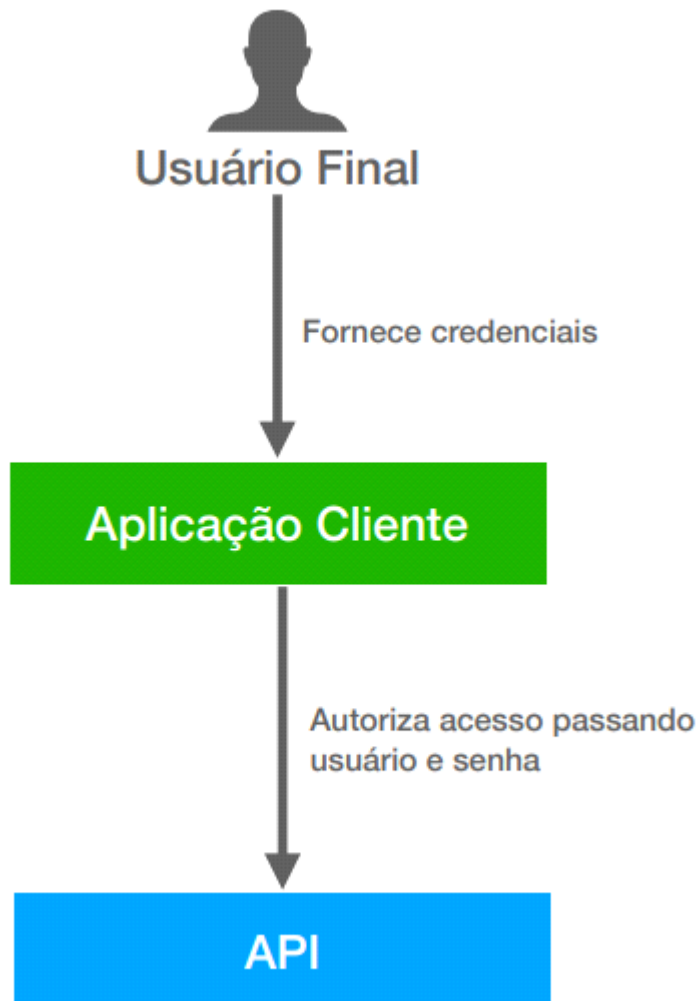
CSRF - Problemas com CSRF: Se a sua aplicação estiver usando proteção CSRF (Cross-Site Request Forgery), a aplicação não irá autorizar um POST por conta que possa estar armazenado a sessão no cookie.

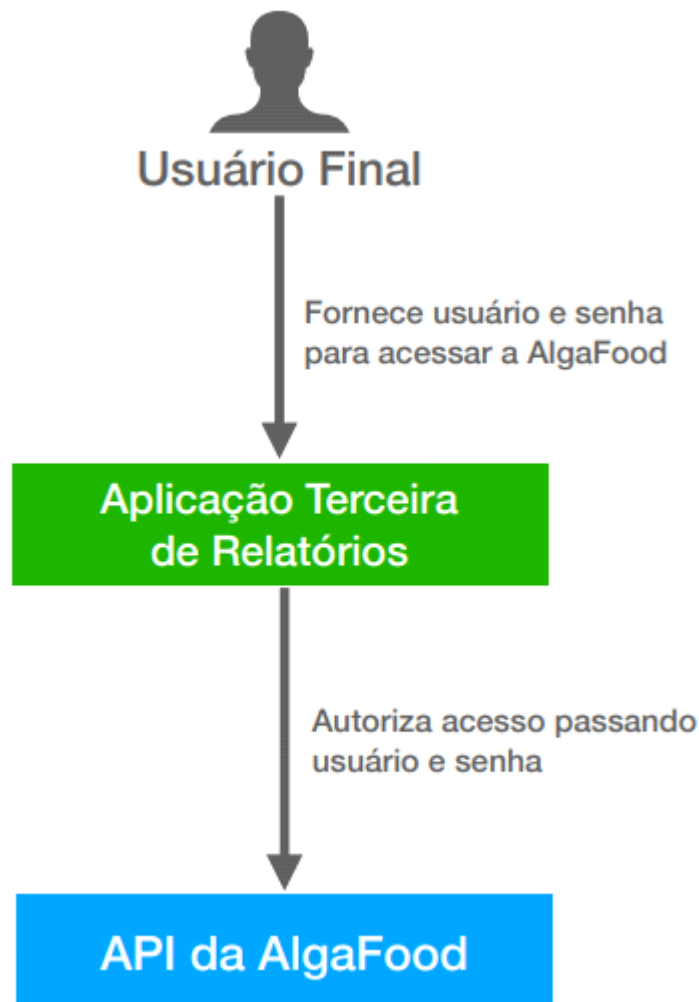
## 22.5. Introdução ao OAuth2

sábado, 29 de abril de 2023 23:45



Modelo tradicional de autorização cliente-servidor:





# O que é OAuth2?

framework de autorização para permitir que aplicações terceiras obtém acessos limitados aos serviços e recursos. Conjunto de regras, protocolo, especificação (que pode ser estendida).

Especificação OAuth2 RFC 6749 <https://datatracker.ietf.org/doc/html/rfc6749>

## The OAuth 2.0 Authorization Framework

### Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in [RFC 5849](#).

### Status of This Memo

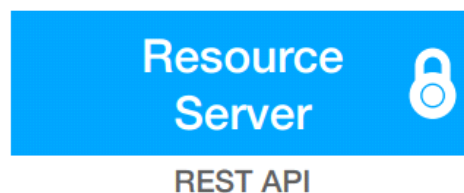
This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in [Section 2 of RFC 5741](#).

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc6749>.

- 4 Papeis (roles) envolvidos na comunicação protocolo

- Resource Server



Servidor que hospeda recursos protegidos. Exemplo da AlgaFood, pedidos, restaurantes, produtos, basicamente a API. Na especificação se chama Resource Server.

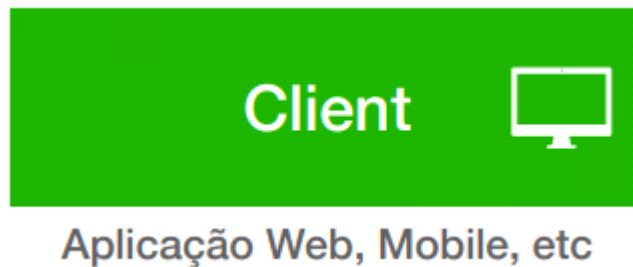
- Resource Owner





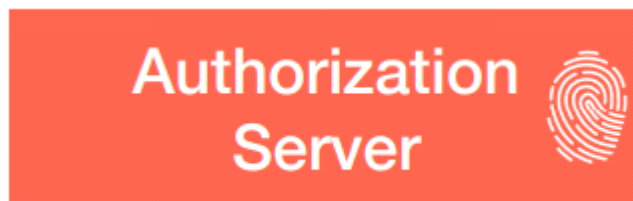
Resource Owner representa o usuário final, o dono do recurso

- Client



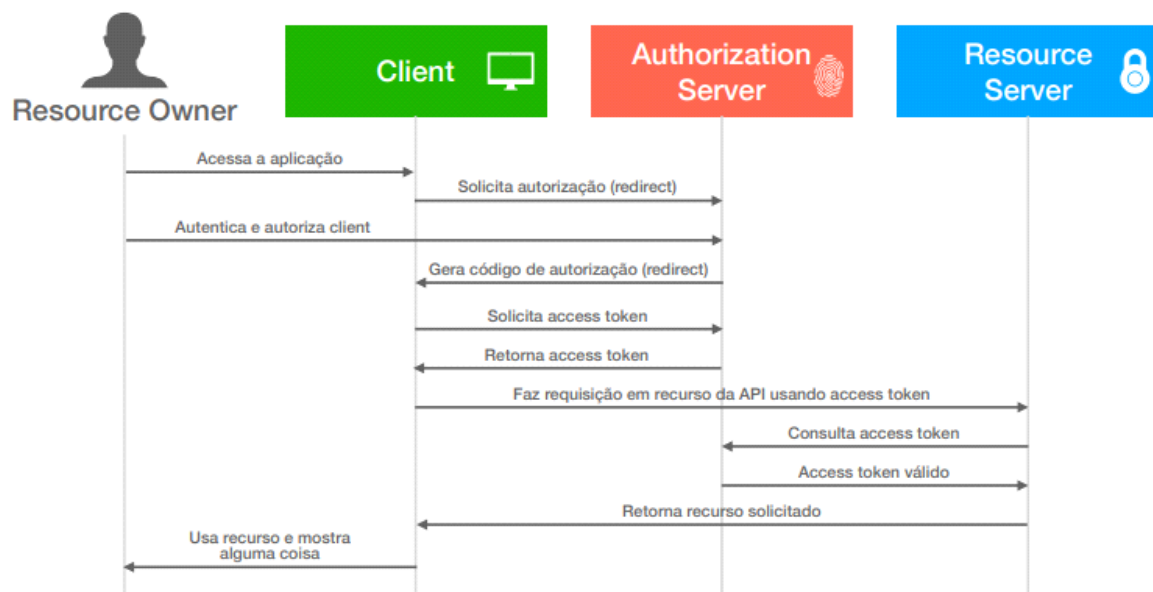
Acessa os recursos protegidos do Resource Server

- Authorization Server



Quem garante o acesso do Resource Owner ao Resource Server . O Authorization Server autentica o Resource Owner e autoriza o Client a obter acesso aos recursos do Resource Server

Fluxo:



Também chamado de Authorization Code Grant ou Flow. Existem outros tipos de fluxos

# 22.6. Soluções para OAuth2: nova stack do Spring Security vs Spring Security Oauth

domingo, 30 de abril de 2023 10:54

## 22.7. Conhecendo o fluxo Resource Owner Password Credentials

domingo, 30 de abril de 2023

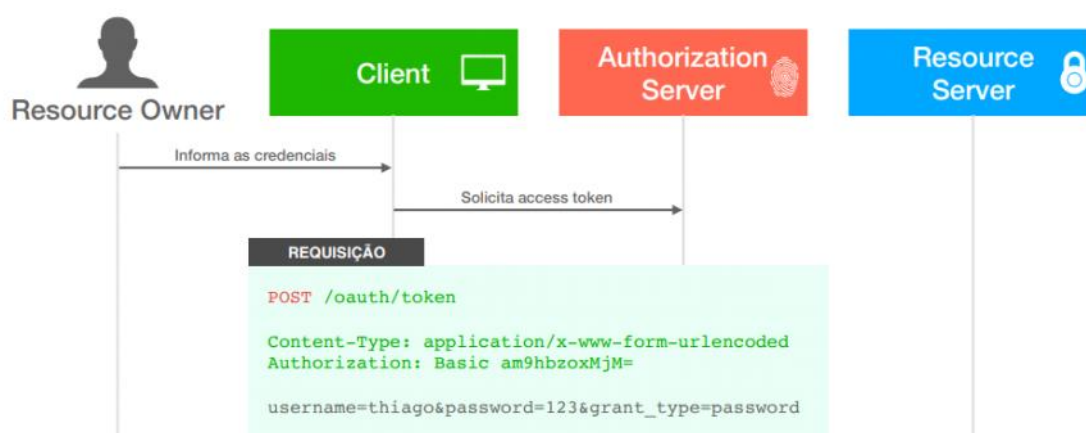
15:38



Tipos de fluxos de autorização ou grant type, é a frequência de comunicação para que o token de acesso seja emitido para o client acessar o resource server.

Resource Owner Password Credentials Grant, Password Credentials, Password Flow: obtém um token de acesso a partir de um usuário e senha, o client oferece o usuário e senha para o authorization server e o mesmo emite o token para o client para que ele possa acessar os recursos do resource server.

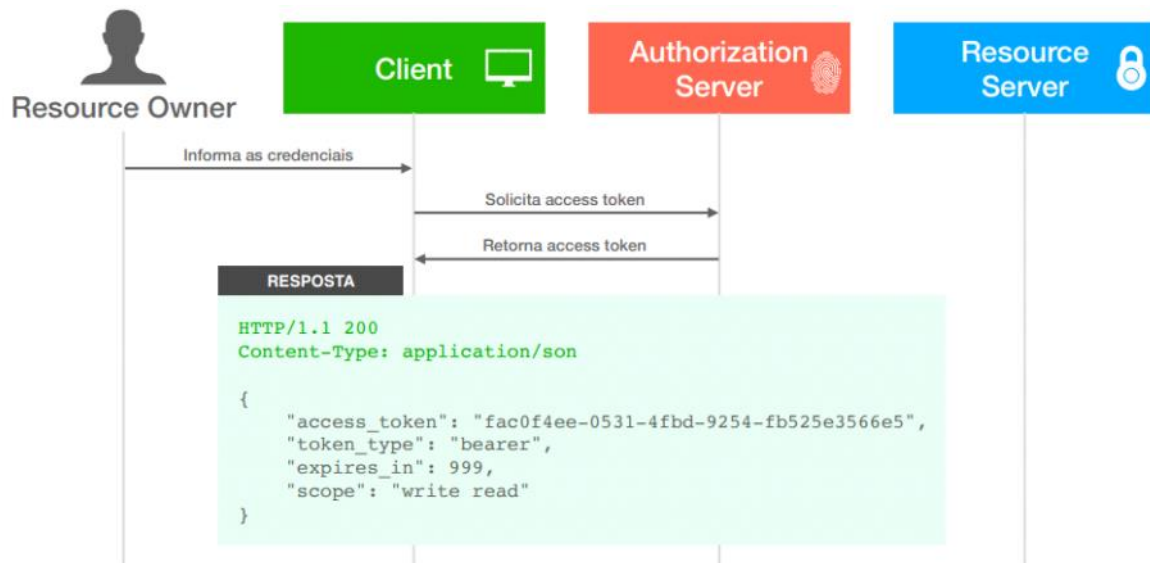
Sequência de comunicação:



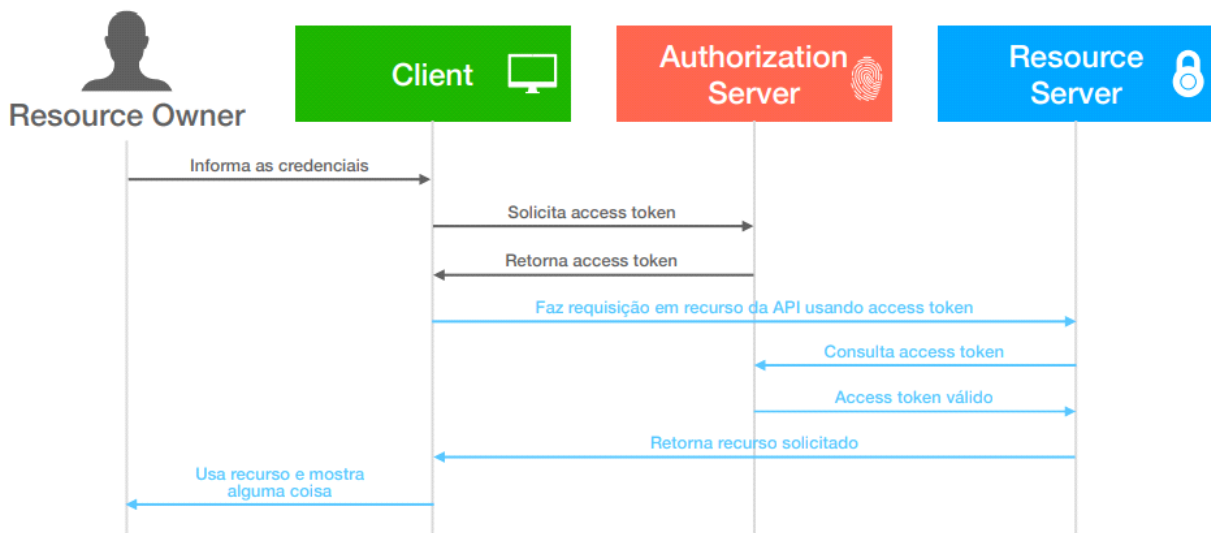
Authorization Basic do próprio client, o client contém um password configurado na aplicação.

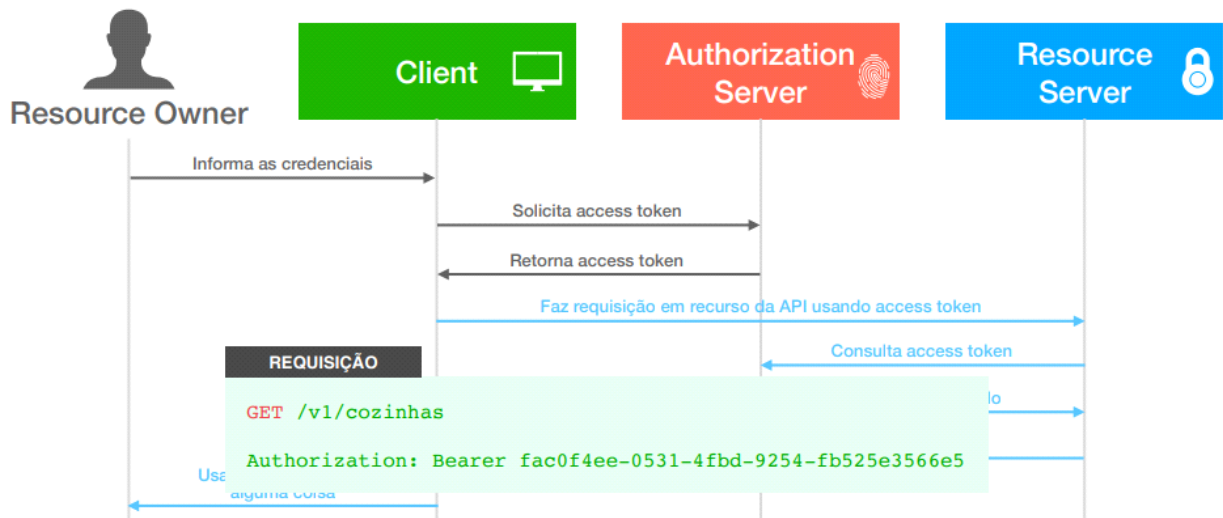
grant\_type é um parâmetro que é usado para especificar o tipo de concessão (ou grant) que está sendo solicitada pelo cliente.

Nesse fluxo, o grant\_type é definido como "password", indicando que o cliente está solicitando um token de acesso usando as credenciais do usuário, ou seja, o nome de usuário (username) e a senha (password).



o cliente não deve armazenar usuário e senha





O cliente tem que ser confiável e além disso, esse fluxo é desencorajado.

## 22.8. Criando o projeto do Authorization Server com Spring Security OAuth2

domingo, 30 de abril de 2023 16:01

Algafood API vai ser o Resource Server e o Authorization Server poderia estar dentro do projeto, ou seja, podemos ter os dois tipos de comunicações no mesmo projeto.

Na aula vamos fazer um Authorization Server separado do projeto atual. Além do mais, ambos os projetos tem que rodar em portas diferentes, para isso, basta configurar a porta no application.properties do novo projeto com server.port=8081

Utilizando a versão do Spring Boot 2.7.11

```
<!--  
https://mvnrepository.com/artifact/org.springframework.security.oauth/spring-security-oauth2 -->  
<dependency>  
  <groupId>org.springframework.security.oauth</groupId>  
  <artifactId>spring-security-oauth2</artifactId>  
  <version>2.5.2.RELEASE</version>  
</dependency>
```

é a última versão antes do desligamento do projeto Spring Security OAuth2, sendo assim, o Spring Security integra os recursos de Authorization Server e Resource Server

```
<!--  
Spring Security OAuth2 com Java 11+ lança exception:  
Caused by: java.lang.ClassNotFoundException: javax.xml.bind.JAXBException  
Em um projeto com Spring Boot, adicione as dependências abaixo para resolver o problema.  
Leia também: https://stackoverflow.com/questions/52502189/java-11-package-javax-xml-bind-does-not-exist  
-->
```

```
-->  
<dependency>  
  <groupId>javax.xml.bind</groupId>  
  <artifactId>jaxb-api</artifactId>  
</dependency>  
<dependency>  
  <groupId>com.sun.xml.bind</groupId>  
  <artifactId>jaxb-core</artifactId>  
  <version>2.3.0.1</version>  
</dependency>  
<dependency>  
  <groupId>com.sun.xml.bind</groupId>  
  <artifactId>jaxb-impl</artifactId>
```

```
<version>${jaxb-jaxb.version}</version>
</dependency>
```

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

}
```

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /*Lista de usuários em memória para autenticação*/
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser( username: "Gustavo")
              .password(passwordEncoder().encode( rawPassword: "123"))
              .roles("ADMIN")
            .and()
            .withUser( username: "Joao")
              .password(passwordEncoder().encode( rawPassword: "123"))
              .roles("ADMIN");
    }

    /*@Override estamos em um "Authorization Server" e essas config são para "Resource Server"*/
    protected void configure(HttpSecurity http) throws Exception {
        /*restrição e autorização de acesso aos endpoints*/
        http.httpBasic()
            .and()
            .authorizeRequests()
            .antMatchers("/v1/cozinhas/**").permitAll()
            .anyRequest().authenticated()

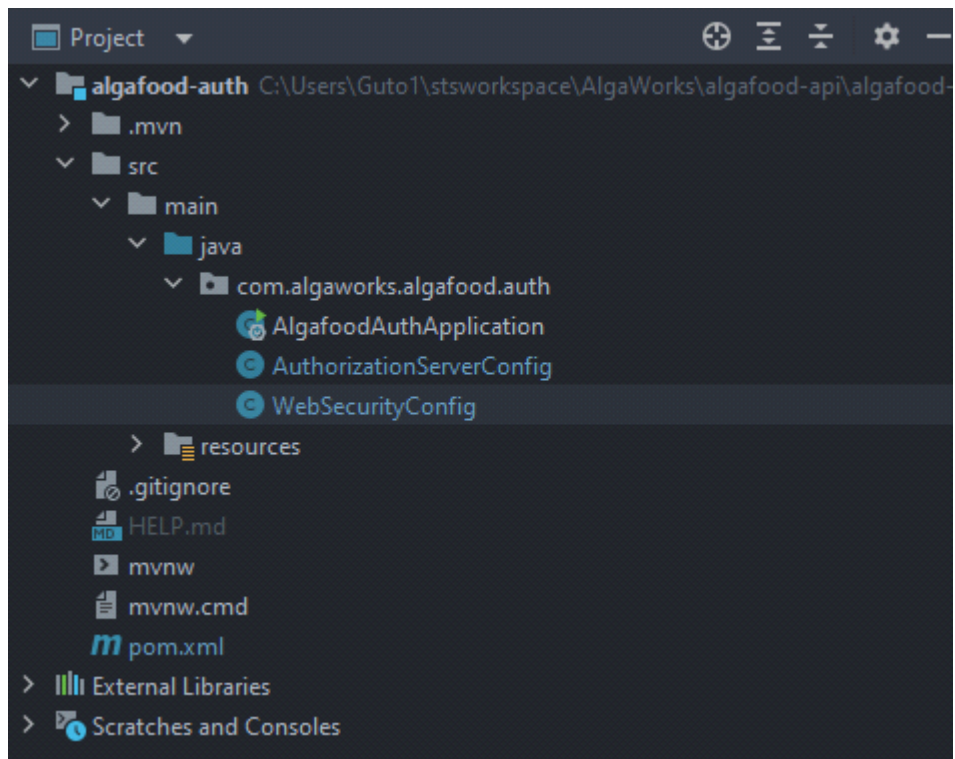
            /*armazenamento de sessão*/
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)

            /*desabilitação de csrf*/
            .and()
            .csrf().disable();
    }

    /*Necessário para criptografar senha*/
    2 usages
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

Um Authorization Server deve conter apenas configurações de autorização.





## 22.9. Configurando o fluxo Authorization Server com Password Credentials e Opaque Tokens

domingo, 30 de abril de 2023 17:10

```
14 @Configuration
15 @EnableAuthorizationServer
16 public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
17
18     1 usage
19     @Autowired
20     private PasswordEncoder passwordEncoder;
21
22     1 usage
23     @Autowired
24     private AuthenticationManager authenticationManager;
25
26     /*configuração de clientes*/
27     @Override
28     public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
29         clients.inMemory() //autorização de clientes em memória
30             .withClient( clientId: "algafood-web" )//cliente consumidor da api
31             .secret(passwordEncoder.encode( rawPassword: "web123" ))//password do client
32             .authorizedGrantTypes("password")//tipo de fluxo Resource Owner Password Credentials GrantType
33             .scopes("write", "read");//escopo de leitura e alteração
34     }
35 }
```

```
34 @Override
35 public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
36     endpoints.authenticationManager(authenticationManager); //somente o fluxo "password" p
37 }
38 }
39 }
```

```
Exception {
password" precisa do authenticationManager, pois é assim que funciona o seu fluxo
```

em WebSecurityConfig

```
/*AuthenticationManager para o Authorization Server, configure (AuthorizationServerEndpointsConfigurer) */
@Bean
@Override
protected AuthenticationManager authenticationManager() throws Exception {
    return super.authenticationManager();
}
```

Estamos utilizando o Fluxo Password

POST ▼ http://localhost:8081/oauth/token

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

Type Basic Auth ▼

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#) ➤

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborator, variables are masked.

Username

Password

POST ▼ http://localhost:8081/oauth/token

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

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

	Key	Value
<input checked="" type="checkbox"/>	username	Gustavo
<input checked="" type="checkbox"/>	password	123
<input checked="" type="checkbox"/>	grant_type	password
	Key	Value

**Body** ▼ Cookies Headers (10) ● Test Results

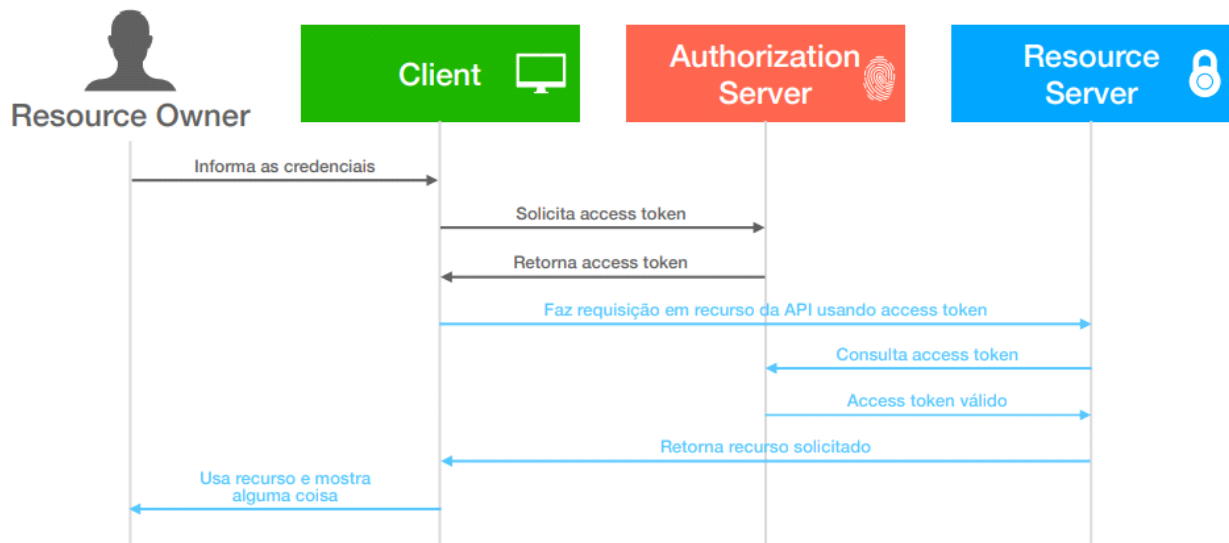
Pretty Raw Preview Visualize JSON ▼ ≡

```

1  {
2    "access_token": "V5A0T0X6hERne03Dumhp0j2Mot8",
3    "token_type": "bearer",
4    "expires_in": 43199,
5    "scope": "write read"
6  }

```

o token de acesso concede ao cliente a autorização no Resource Server



para aumentar o tempo de token válido

```

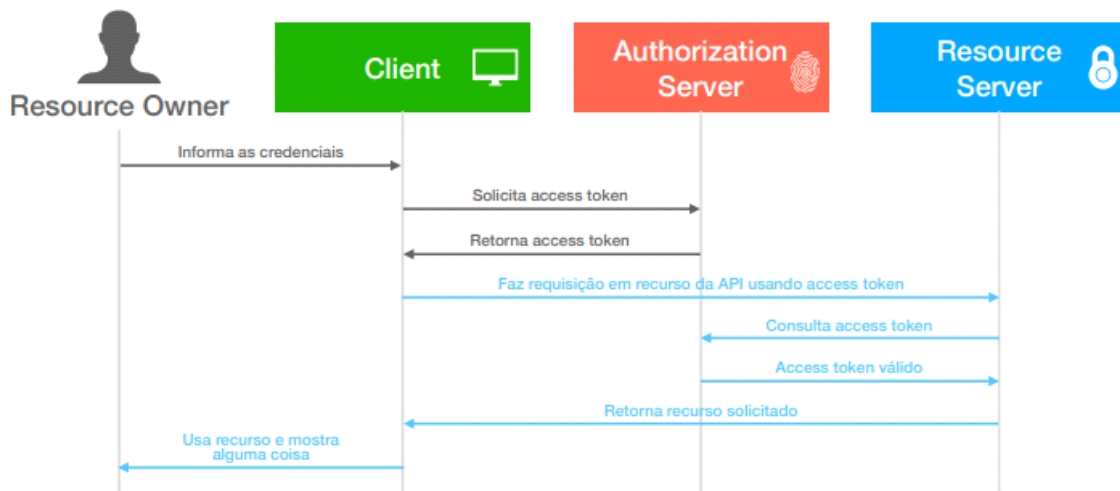
@Autowired
private AuthenticationManager authenticationManager;

/*configuração de clientes*/
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory() //autorização de clientes em memória
        .withClient( clientId: "algafood-web" ) //cliente consumidor da api
        .secret( passwordEncoder.encode( rawPassword: "web123" ) ) //password do client
        .authorizedGrantTypes( "password" ) //tipo de fluxo Resource Owner Password Credentials GrantType
        .scopes( "write", "read" ) //escopo de leitura e alteração
        .accessTokenValiditySeconds( 60 * 60 * 6 ); //equivale a 6 horas
}
  
```

## 22.10. Configurando o endpoint de introspecção de tokens no Authorization Server

domingo, 30 de abril de 2023 18:51

Quando o cliente recebe o access token e faz uma requisição no recurso no Resource Server, o Resource Server faz uma consulta de validação do token para o Authorization Server, se válido, o Resource Server disponibiliza o recurso para o cliente, caso contrário, emite um status de não autorizado



A especificação do OAuth2 não menciona como a consulta do token para o Authorization Server deve funcionar, mas há outra especificação que padroniza isso. A RFC7662 <https://datatracker.ietf.org/doc/html/rfc7662>

## OAuth 2.0 Token Introspection

### Abstract

This specification defines a method for a protected resource to query an OAuth 2.0 authorization server to determine the active state of an OAuth 2.0 token and to determine meta-information about this token. OAuth 2.0 deployments can use this method to convey information about the authorization context of the token from the authorization server to the protected resource.

### Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in [Section 2 of RFC 5741](#).

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7662>.

### Copyright Notice

A RFC padroniza o endpoint de introspecção de token. O processo de checagem de token é chamado de introspecção. O projeto Spring Security OAuth2 não segue à risca a especificação, pois a especificação é mais recente que o projeto. Porém, é bem parecido.

sobrescrevendo o método  
`configure(AuthorizationServerSecurityConfigurer security)`

```

/*configuração de clientes*/
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory() //autorização de clientes em memória
        .withClient( clientId: "algafood-web")//cliente consumidor da api
        .secret(passwordEncoder.encode( rawPassword: "web123"))//password do client
        .authorizedGrantTypes("password")//tipo de fluxo Resource Owner Password Credentials GrantType
        .scopes("write", "read")//escopo de leitura e alteração
        .accessTokenValiditySeconds(60 * 60 * 6); //equivale a 6 horas
}

/*Para configurar o acesso ao endpoint de checagem de token ou check token, instrospecção de token */
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    security.checkTokenAccess("isAuthenticated()");//Spring Security Expression.
    // P/ acessar o check token, precisa de autenticação do cliente
}

@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints.authenticationManager(authenticationManager); //somente o fluxo "password" precisa do authenticationM
}

```

utilizando Expressão do Spring Security

Primeiro chamamos o endpoint de geração de token, passando um Basic Authorization

POST

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

Type

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a colla variables

Username

Password

e um body

	Key	Value
<input checked="" type="checkbox"/>	username	Gustavo
<input checked="" type="checkbox"/>	password	123
<input checked="" type="checkbox"/>	grant_type	password
	Key	Value

```
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "access_token": "mh7j082An_jZDKuk86tVlMacjdk",
3   "token_type": "bearer",
4   "expires_in": 21470,
5   "scope": "write read"
6 }
```

agora chamamos o endpoint de checagem de token oauth/check\_token

POST http://localhost:8081/oauth/check\_token

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

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

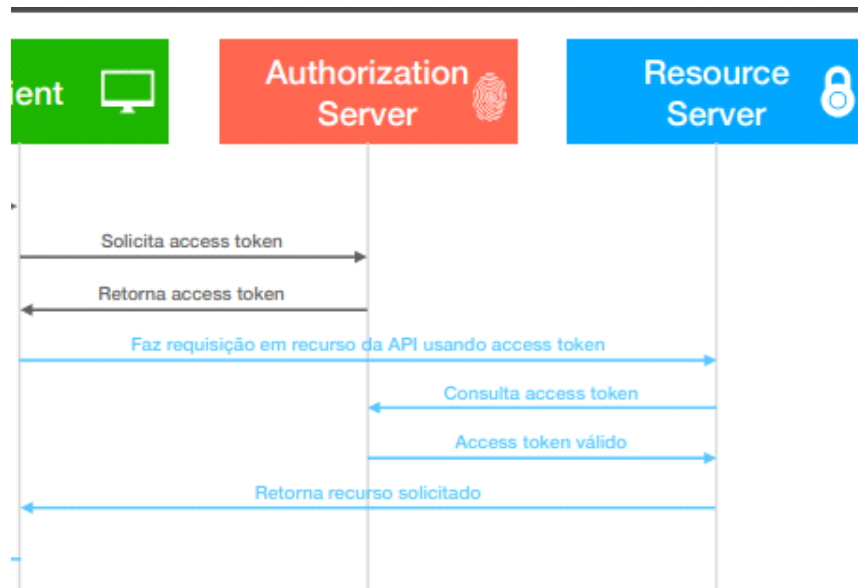
	Key	Value
<input type="checkbox"/>	username	Gustavo
<input type="checkbox"/>	password	123
<input type="checkbox"/>	grant_type	password
<input checked="" type="checkbox"/>	token	mh7j082An_jZDKuk86tVIMacjdk
	Key	Value

passando um body com x-www-form-urlencoded junto com o Basic Authorization

```
Body Cookies Headers (11) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "active": true,
3   "exp": 1682917269,
4   "user_name": "Gustavo",
5   "authorities": [
6     "ROLE_ADMIN"
7   ],
8   "client_id": "algafood-web",
9   "scope": [
10    "write",
11    "read"
12  ]
13 }
```

Dessa forma, o Resource Server pode consultar o Authorization Server com a checagem de token, chamando o endpoint





Também podemos autorizar o cliente somente através do endpoint de introspecção de token, permitindo todos usando `security.checkTokenAccess("PermitAll()")`

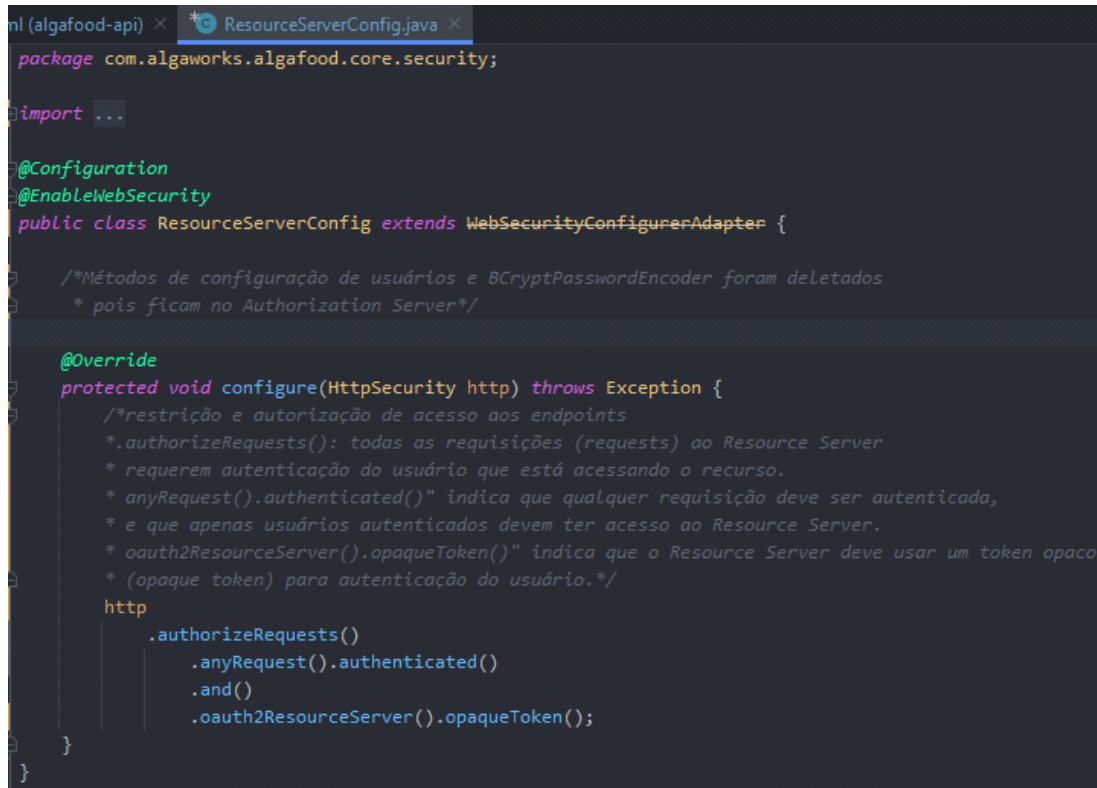
```
/*Para configurar o acesso ao endpoint de checagem de token ou check token, introspecção de token*/
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    //security.checkTokenAccess("isAuthenticated()");//Spring Security Expression.
    security.checkTokenAccess("PermitAll()");//permitindo qualquer cliente
    // P/ acessar o check token, precisa de autenticação do cliente
}
```

## 22.11. Configurando o Resource Server com a nova stack do Spring Security

domingo, 30 de abril de 2023 20:44

Utilizando o Spring Security OAuth2 para configurar Resource Server

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```



```
package com.algaworks.algafood.core.security;

import ...

@Configuration
@EnableWebSecurity
public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

    /*Métodos de configuração de usuários e BCryptPasswordEncoder foram deletados
    * pois ficam no Authorization Server*/

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        /*restrição e autorização de acesso aos endpoints
        *.authorizeRequests(): todas as requisições (requests) ao Resource Server
        * requerem autenticação do usuário que está acessando o recurso.
        * anyRequest().authenticated() indica que qualquer requisição deve ser autenticada,
        * e que apenas usuários autenticados devem ter acesso ao Resource Server.
        * oauth2ResourceServer().opaqueToken() indica que o Resource Server deve usar um token opaco
        * (opaque token) para autenticação do usuário.*/
        http
            .authorizeRequests()
                .anyRequest().authenticated()
                .and()
                .oauth2ResourceServer().opaqueToken();
    }
}
```

O método "configure" está sendo usado para definir a configuração de segurança do recurso (Resource Server) do aplicativo usando o framework Spring Security e o protocolo OAuth2.

A chamada "authorizeRequests()" define que todas as requisições (requests) ao Resource Server requerem autenticação do usuário que está acessando o recurso.

A chamada "anyRequest().authenticated()" indica que qualquer requisição deve ser autenticada, e que apenas usuários autenticados devem ter acesso ao Resource Server.

Por fim, a chamada "oauth2ResourceServer().opaqueToken()" indica que o Resource Server deve usar um token opaco (opaque token) para autenticação do usuário.

Em resumo, esse código está definindo a configuração básica de segurança para um Resource Server em um aplicativo web, especificando que todos os acessos a esse recurso devem ser autenticados e que o token a ser usado para autenticação é opaco.

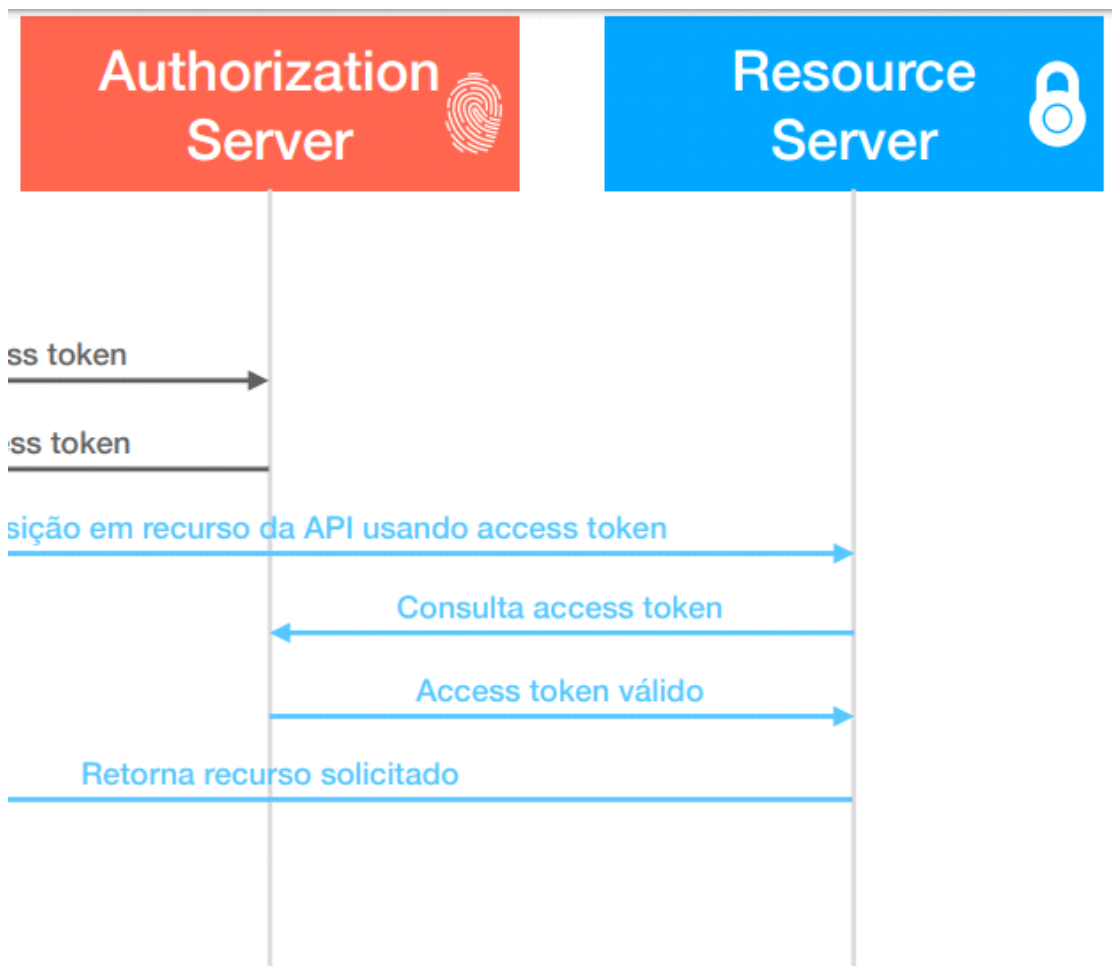
O opaque token, ou token opaco em português, é um tipo de token de autenticação que é emitido pelo provedor de autenticação (Authorization Server) no protocolo OAuth 2.0.

No fluxo de autorização OAuth 2.0, quando um cliente solicita acesso a um recurso protegido, ele é redirecionado para o provedor de autenticação, onde o usuário é autenticado e um token de acesso é gerado. Esse token pode ser do tipo JWT (JSON Web Token), que contém informações sobre o usuário e outras informações de autorização, ou pode ser do tipo opaco.

No caso do token opaco, ele é gerado pelo provedor de autenticação e armazenado em seu banco de dados. Quando um cliente faz uma solicitação ao recurso protegido, ele apresenta o token opaco ao Resource Server, que envia uma solicitação ao provedor de autenticação para verificar a validade do token e autorizar o acesso ao recurso.

Portanto, o provedor de autenticação é responsável pela emissão e validação do token opaco. O Resource Server não tem acesso ao conteúdo do token e não pode verificar sua validade diretamente, sendo necessário enviar uma solicitação ao provedor de autenticação para fazer essa verificação.

Agora precisamos configurar o Resource Server para fazer a validação no Authorization Server do token do cliente. Já temos o endpoint de introspecção configurado no Authorization Server da aula [22.10. Configurando o endpoint de introspecção de tokens no Authorization Server](#) agora teremos que chamar o endpoint a partir do Resource Server



Nossa api é o Resource Server e o projeto AlgaFood-auth é o Authorization Server

Configurando a comunicação do Resource para o Authorization

```
#22.11. Configurando o Resource Server com a nova stack do Spring Security
#configurando a comunicação do Resource Server para o Authorization Server
spring.security.oauth2.resourceserver.opaquetoken.introspection-uri=http://localhost:8081/oauth/check_token
spring.security.oauth2.resourceserver.opaquetoken.client-id=algafood-web
spring.security.oauth2.resourceserver.opaquetoken.client-secret=web123
```

Agora ao fazer uma requisição, temos primeiro que gerar um access token através do cliente, informando informações do usuário e as credenciais do cliente

POST http://localhost:8081/oauth/token

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

Type Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Username algafood-web

Password web123

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "access_token": "9-JHKgRUVy2q8ROvVbzRyydbIGk",
3   "token_type": "bearer",
4   "expires_in": 21599,
5   "scope": "write read"
6 }
```

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

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

Key	Value
<input checked="" type="checkbox"/> username	Gustavo
<input checked="" type="checkbox"/> password	123
<input checked="" type="checkbox"/> grant_type	password
Key	Value

Agora temos acesso aos recursos informando o token de acesso no cabeçalho da requisição

GET

http://localhost:8080/v1/cozinhas?size=1&page=0&sort=nome

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

☒

Postman-Token

<calculated when request is sent>

☒

Host

<calculated when request is sent>

☒

User-Agent

PostmanRuntime/7.32.2

☒

Accept

\*/\*

☒

Accept-Encoding

gzip, deflate, br

☒

Connection

keep-alive

☐

Accept

application/xml

☒

Authorization

Bearer 9-JHKgRUVy2q8ROvVbzRyydbIGk

Key

Value

```

/*configuração de clientes*/
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory() //autorização de clientes em memória
        .withClient( clientId: "algafood-web")//cliente consumidor da api
        .secret(passwordEncoder.encode( rawPassword: "web123"))//password do client
        .authorizedGrantTypes("password")//tipo de fluxo Resource Owner Password Credentials GrantType
        .scopes("write", "read")//escopo de leitura e alteração
        .accessTokenValiditySeconds(60 * 60 * 6)//equivale a 6 horas
        .and()
        .withClient( clientId: "checktoken")/*Acesso somente para verificar o token no ResourceServer*/
        .secret("check123");
}

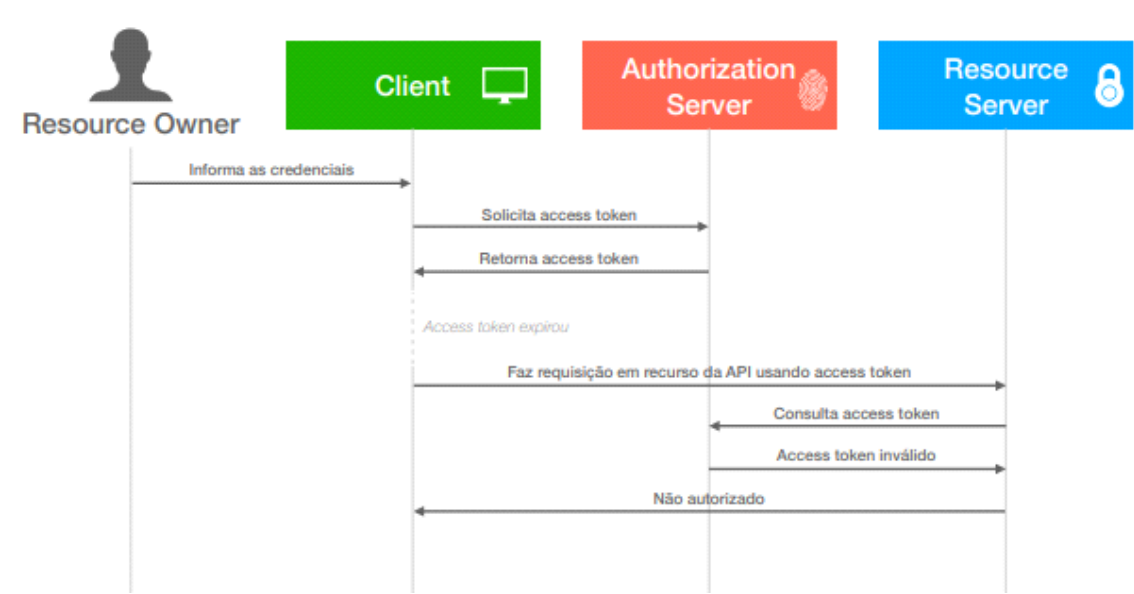
```

vamos adicionar mais um client que faz o papel de client resource server para a checagem de token no Authorization Server. O algafood-web é o cliente que tem o acesso no Authorization Server e o Checktoken é o "cliente" que acessa a checagem do token no do Resource Server para o Authorization Server

# 22.12. Conhecendo o fluxo para emitir e usar Refresh Tokens

terça-feira, 2 de maio de 2023 07:48

- E se o Token expirar?



retornando um status 401

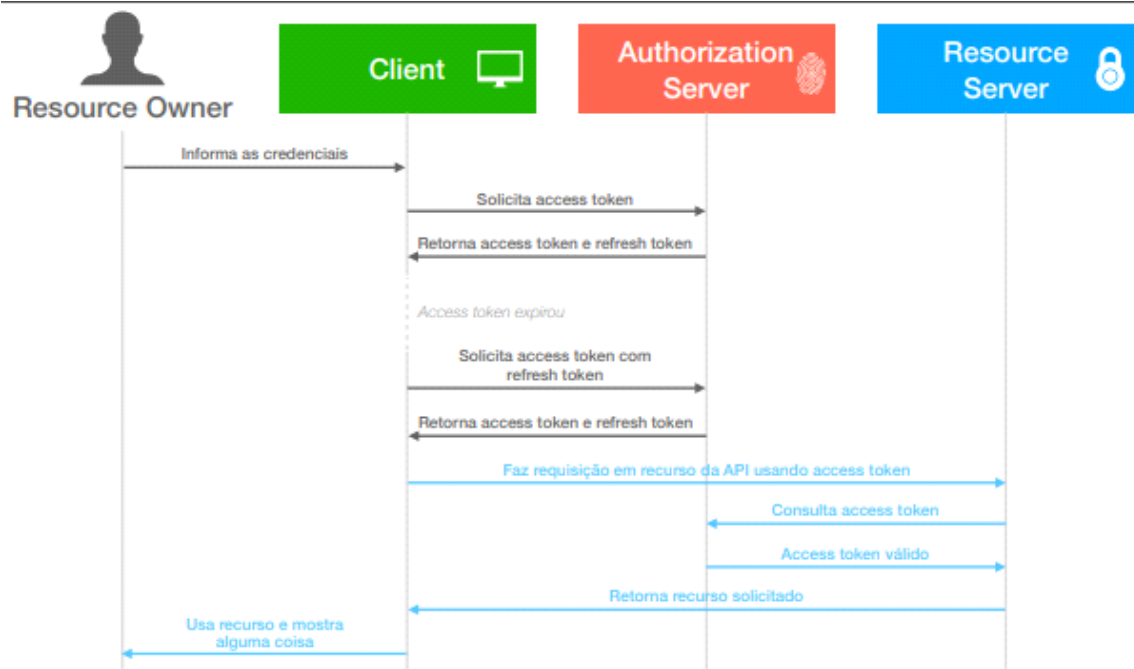
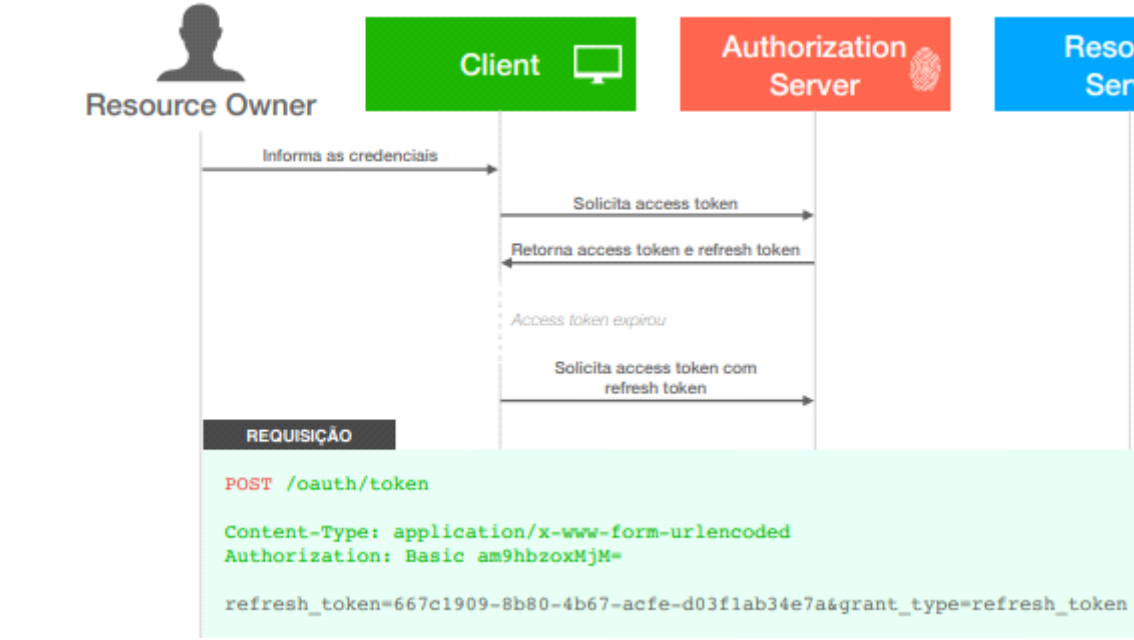
- Emitindo e usando Refresh Tokens

O Authorization Serve pode emitir dois tipos de Tokens, o Access e o Refresh, para quando o Access expirar, temos o Refresh Token para emitir novamente um Access Token, logicamente o Refresh Token terá um tempo de expiração maior para ser utilizado na renovação do Access, permitindo que o Client tenha possibilidade de sempre ter outro Access Token sempre que expirado, sem que o Resource Owner seja solicitado.



retorna dois tipos de Tokens, com o Refresh Token com um objetivo bem definido.

Quando o Access Token estiver inválido, o Client solicita um novo Access Token





# 22.13. Configurando o Refresh Token Grant Type no Authorization Server

terça-feira, 2 de maio de 2023 07:59

Vamos configurar o Auth Server para a emissão de Refresh Token também, por padrão o Refresh Token tem a duração de 30 dias

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

    1 usage
    @Autowired
    private PasswordEncoder passwordEncoder;

    1 usage
    @Autowired
    private AuthenticationManager authenticationManager;

    /*configuração de clientes*/
    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        clients.inMemory() //autorização de clientes em memória
            .withClient( clientId: "algafood-web")//cliente consumidor da api
            .secret(passwordEncoder.encode( rawPassword: "web123"))//password do client
            .authorizedGrantTypes("password")//tipo de fluxo Resource Owner Password Credentials GrantType
            .scopes("write", "read")//escopo de leitura e alteração
            .accessTokenValiditySeconds(60 * 60 * 6)//equivale a 6 horas
            .and()
            .withClient( clientId: "checktoken")/*Acesso somente para verificar o token no ResourceServer*/
            .secret("check123");
    }

    /*Para configurar o acesso ao endpoint de checagem de token ou check token, introspecção de token */
    @Override
    public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
        //security.checkTokenAccess("isAuthenticated()");//Spring Security Expression.
        security.checkTokenAccess("permitAll()");//permitindo qualquer cliente
        // P/ acessar o check token, precisa de autenticação do cliente
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.authenticationManager(authenticationManager); //somente o fluxo "password" precisa do authenti
    }
}
```

adicionado refresh\_token no authorizedGrantType

```
.secret(passwordEncoder.encode( rawPassword: "web123"))//p
.authorizedGrantTypes("password", "refresh_token")//tipo
.scopes("write", "read")//escopo de leitura e alteração
```

AlgaFood - Authorization Server / Refresh\_Token

POST

http://localhost:8081/oauth/token

Params

Authorization

Headers (9)

Body

Pre-request Scr

none

form-data

x-www-form-urlencoded

raw

binary

	Key
<input checked="" type="checkbox"/>	refresh_token
<input type="checkbox"/>	password
<input checked="" type="checkbox"/>	grant_type
	Key

Body

Cookies

Headers (10)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "access_token": "vsxr-qH8uh95ioJgalPhVNHZdjM",
3    "token_type": "bearer",
4    "refresh_token": "vQ7Gx1b3BSvMEctY13nouGQZLUE",
5    "expires_in": 21599,
6    "scope": "write read"
7  }
```

Na classe AuthorizationServerConfig no projeto do AuthorizationServer

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints.authenticationManager(authenticationManager); //somente o fluxo "password" p
    endpoints.userDetailsService(userDetailsService); /*para refresh_token*/
}
}
```

Na classe WebSecurityConfig

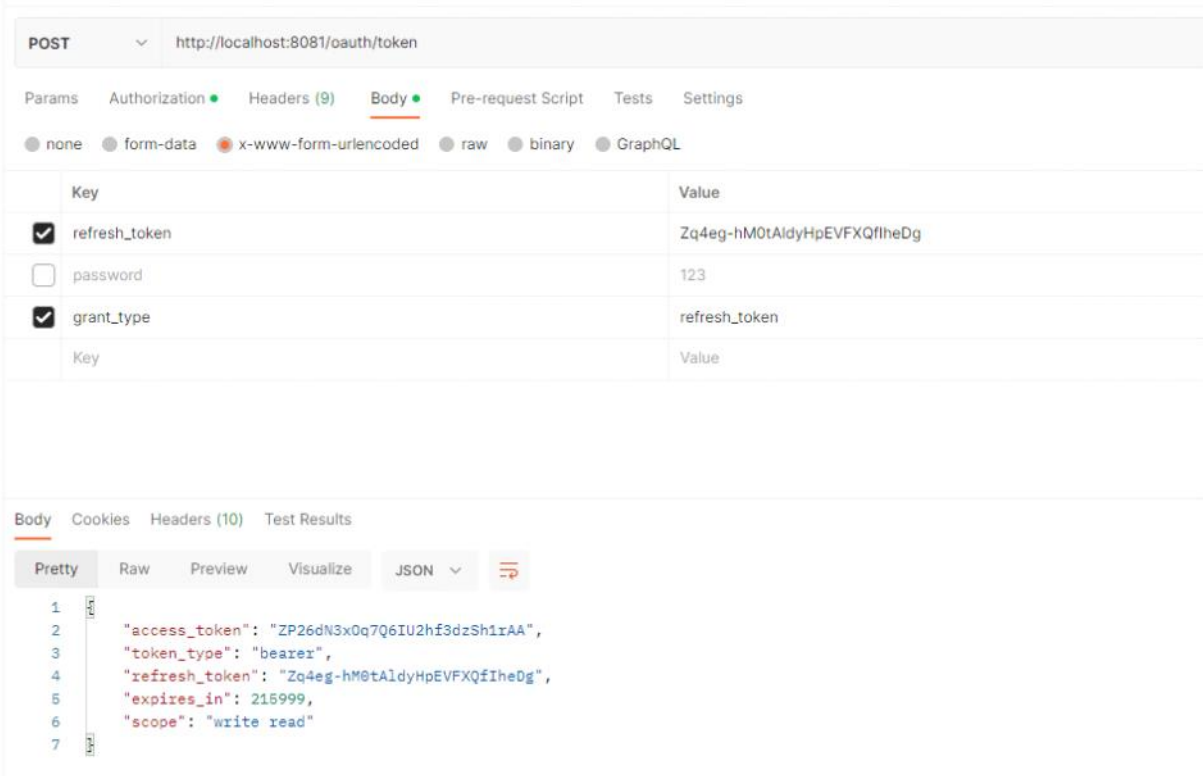
```
/*User Details Service para o Refresh Token*/
@Bean
@Override
protected UserDetailsService userDetailsService() {
    return super.userDetailsService();
}
```

# 22.14. Configurando a validade e não reutilização de Refresh Tokens

sexta-feira, 5 de maio de 2023 20:43

```
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory() //autorização de clientes em memória
        .withClient( clientId: "algafood-web")//cliente consumidor da api
        .secret(passwordEncoder.encode( rawPassword: "web123"))//password do client
        .authorizedGrantTypes("password", "refresh_token")//tipo de fluxo Resource Owner Password Credentials
        .scopes("write", "read")//escopo de leitura e alteração
        .accessTokenValiditySeconds(60 * 60 * 60)//equivale a 6 horas
        .refreshTokenValiditySeconds(60 * 24 * 60 * 60)//60 dias * 24 horas * 60m * 60s
        .and()
        .withClient( clientId: "checktoken")/*Acesso somente para verificar o token no ResourceServer*/
        .secret("check123");
}
```

Ao atualizar o Token de Acesso quando o mesmo expira



o refresh\_token não é atualizado também, ou seja, o refresh token ainda permanece igual até que o tempo de expiração dele seja inválido. para alterar isso, podemos usar `reuseRefreshTokens`

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints.authenticationManager(authenticationManager)
        .userDetailsService(userDetailsService)
        .reuseRefreshTokens(false);
}
```

agora sempre que usamos o refresh token para atualizar o token de acesso, o refresh token também irá alterar



## 22.15. Conhecendo o fluxo Client Credentials

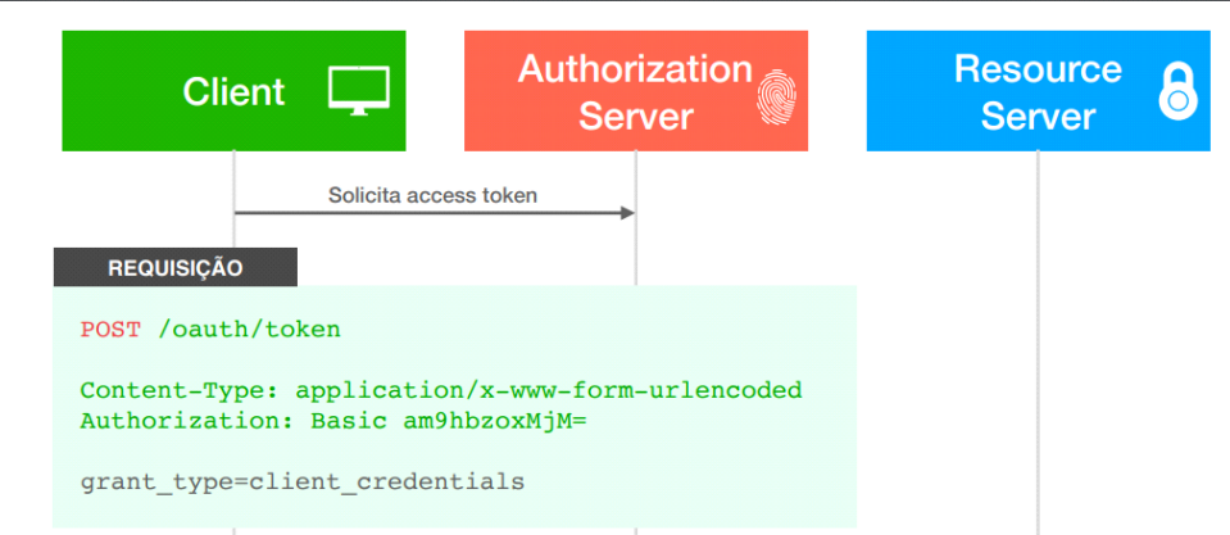
sexta-feira, 5 de maio de 2023 21:22



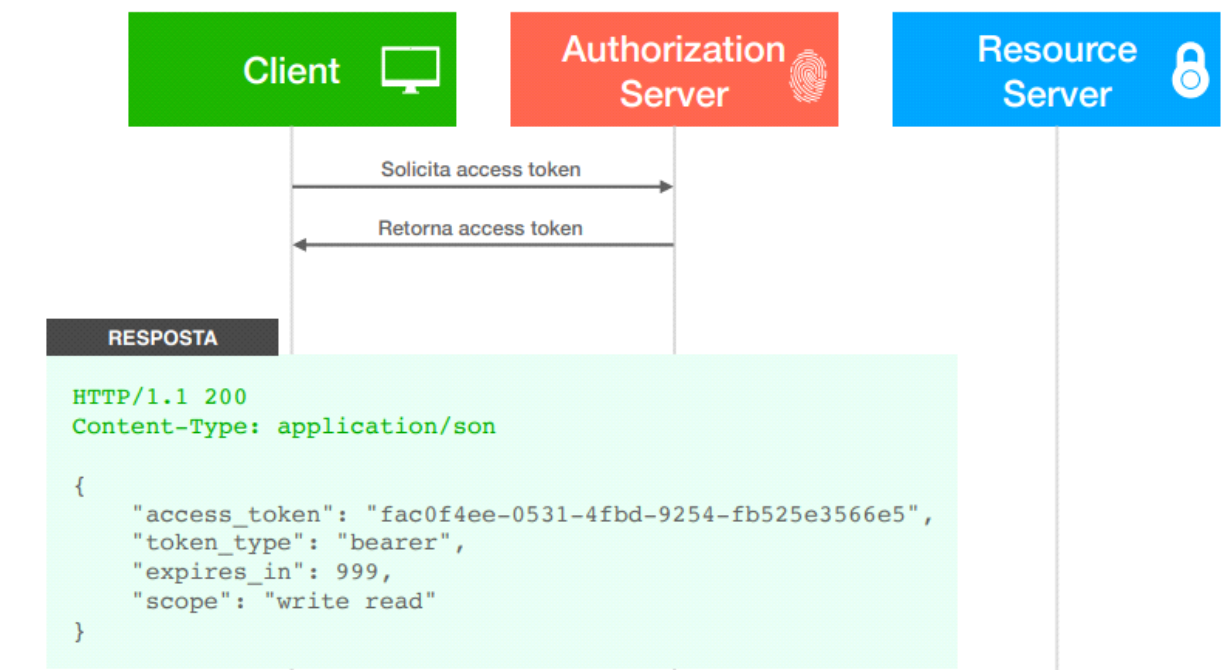
O Access Token que o Authorization Server emite, é uma chave de acesso ao Resource Server para o usuário final, o Resource Owner.

Imagine uma aplicação puramente backend, que não tem interação com o usuário e que precisa acessar o resource server, logicamente, não precisa gerar token a um usuário específico

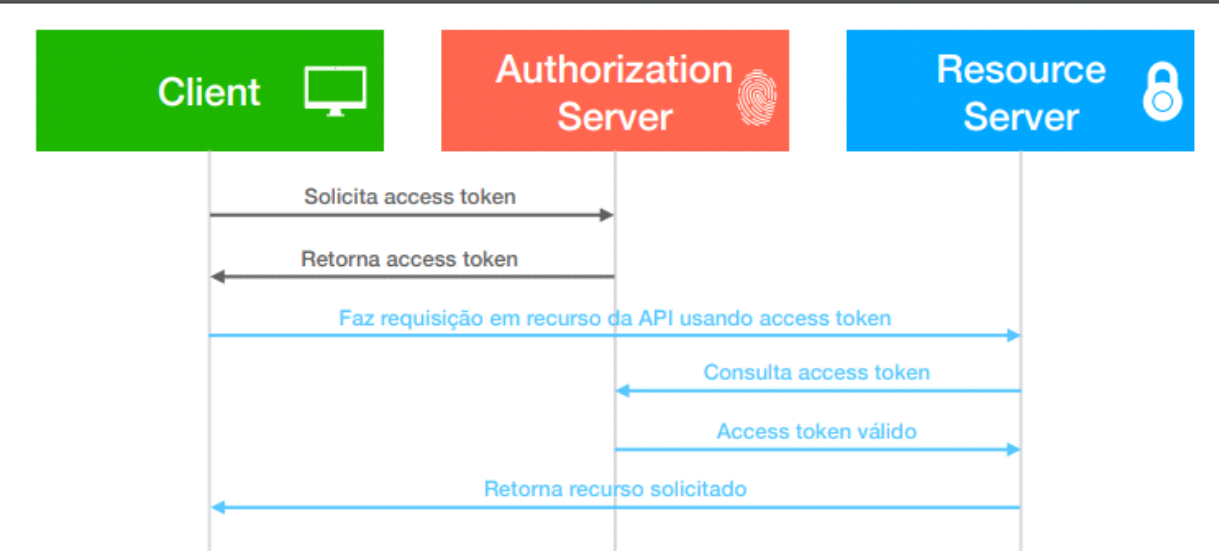
O fluxo Client Credentials Grant garante acesso somente com as credenciais do cliente



Diferente do Password Flow, o Cliente se autentica com o Authorization Server passando somente as credenciais o grant type = client\_credentials



Não utiliza refresh token



## 22.16. Configurando o Client Credentials Grant Type no Authorization Server

sexta-feira, 5 de maio de 2023

21:31

```
/*configuração de clientes*/
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.inMemory() //autorização de clientes em memória
        .withClient("algafood-web")//cliente consumidor da api
        .secret(passwordEncoder.encode("web123"))//password do client
        .authorizedGrantTypes("password", "refresh_token")//tipo de fluxo Resource Owner Password Credentials GrantType
        .scopes("write", "read")//escopo de leitura e alteração
        .accessTokenValiditySeconds(60 * 60 * 60)//equivale a 6 horas
        .refreshTokenValiditySeconds(60 * 24 * 60 * 60)//60 dias * 24 horas * 60m * 60s
        .and()/*client credentials grant type*/
        .withClient("faturamento")
        .secret(passwordEncoder.encode("faturamento123"))//password do client
        .authorizedGrantTypes("client_credentials")//tipo de fluxo Resource Owner Password Credentials GrantType
        .scopes("write", "read")//escopo de leitura e alteração
        .and()
}
```

AlgaFood - Authorization Server / Client Credentials - Acess Token

POST  http://localhost:8081/oauth/token

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

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







	Key	Value
<input type="checkbox"/>	username	Gustavo
<input type="checkbox"/>	password	123
<input checked="" type="checkbox"/>	grant_type	password
	Key	Value

Não precisamos do username e do password (Resource Owner) e passamos "client\_credentials" no grant\_type

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

	Key	Value
<input type="checkbox"/>	username	Gustavo
<input type="checkbox"/>	password	123
<input checked="" type="checkbox"/>	grant_type	client_credentials
	Key	Value

e agora é necessário autenticar o client

POST  http://localhost:8081/oauth/tokenParams Authorization  Headers (9) Body  Pre-request Script none  form-data  x-www-form-urlencoded  raw  binary

Key

☐

username

☐

password

☒

grant\_type

Key

Body  Cookies Headers (10) Test Results

Pretty

Raw

Preview

Visualize

JSON 

```
1  {
2    "access_token": "-Ipr510DrNdeoLC9TSrql0EC3_Q",
3    "token_type": "bearer",
4    "expires_in": 43199,
5    "scope": "write read"
6  }
```

POST

▼

http://localhost:8081/oauth/check\_token

ParamsAuthorizationHeaders (8)Body ●Pre-request ScriptTestsSettings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary


● GraphQL

	Key	Value
<input type="checkbox"/>	username	Gustavo
<input type="checkbox"/>	password	123
<input type="checkbox"/>	grant_type	password
<input checked="" type="checkbox"/>	token	-lpr51ODrNdeoLC9TSrqlOEC3_Q
	Key	Value

BodyCookiesHeaders (11)Test Results

PrettyRawPreviewVisualizeJSON ▼

```
1 {
2   "scope": [
3     "write",
4     "read"
5   ],
6   "active": true,
7   "exp": 1683386380,
8   "client_id": "faturamento"
9 }
```



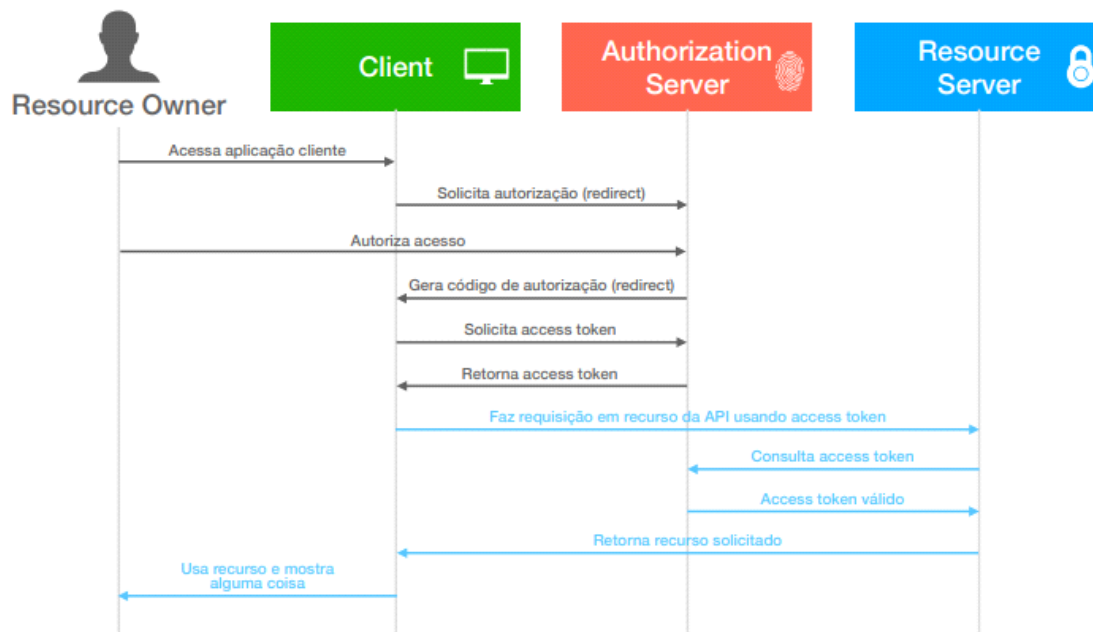
# Authorization Code Grant

## 22.18. Configurando o Authorization Code Grant Type

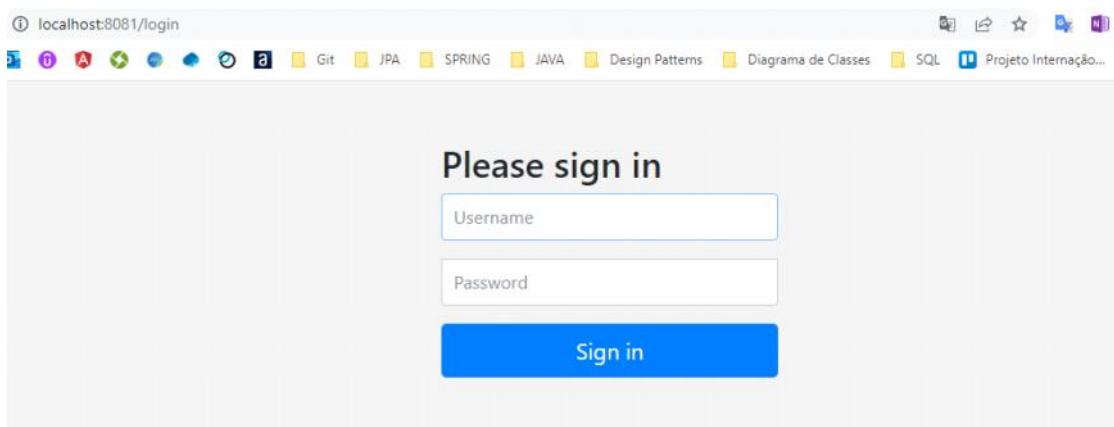
segunda-feira, 8 de maio de 2023 21:50

```
.and()
    .withClient( clientId: "checktoken")/*Acesso somente para verificar o token
    .secret("check123")
    .and()
    .withClient( clientId: "foodanalitycs")
    .secret("food123")
    .authorizedGrantTypes("authorization_code")
    .scopes("write", "read")
    .redirectUri( ...registeredRedirectUri: "http://aplicacao-clientes");
```

[http://localhost:8081/oauth/authorize?response\\_type=code&client\\_id=foodanalitycs&state=abc&redirect\\_uri=http://aplicacao-clientes](http://localhost:8081/oauth/authorize?response_type=code&client_id=foodanalitycs&state=abc&redirect_uri=http://aplicacao-clientes)



equivale ao cliente clicar em um link de login e irá redirecionar para o Authorization Server



```

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /*Lista de usuários em memória para autenticação*/
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser( username: "Gustavo")
              .password(passwordEncoder().encode( rawPassword: "123"))
              .roles("ADMIN")
            .and()
            .withUser( username: "Joao")
              .password(passwordEncoder().encode( rawPassword: "123"))
              .roles("ADMIN");
    }
}

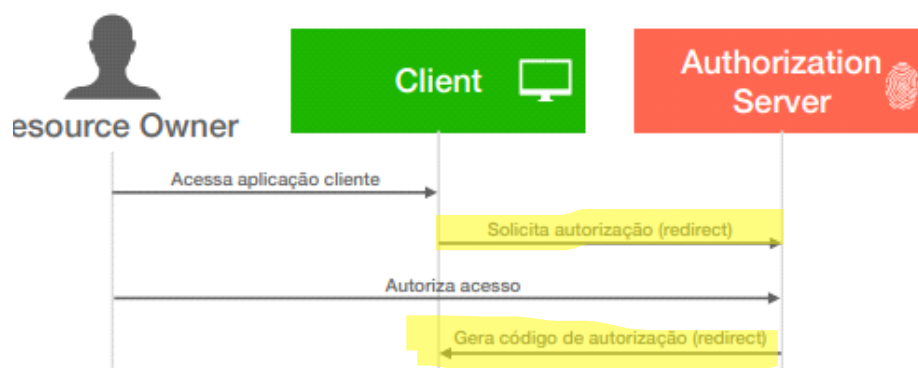
```

## OAuth Approval

Do you authorize "foodanalytics" to access your protected resources?

- scope.write: ☐ Approve ☒ Deny
- scope.read: ☐ Approve ☒ Deny

Authorize



Equivale a geração de código a partir do Auth. Server

e com o code conseguimos solicitar o Access Token

POST ▼ http://localhost:8081/oauth/token

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

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

	Key	Value
<input checked="" type="checkbox"/>	grant_type	authorization_code
<input checked="" type="checkbox"/>	code	SuKzpp
<input checked="" type="checkbox"/>	redirect_uri	http://aplicacao-clientes
	Key	Value

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {  
2   "access_token": "u9sVn1C_Vb4t-UA0w7hVecEBMqE",  
3   "token_type": "bearer",  
4   "expires_in": 43199,  
5   "scope": "read write"  
6 }
```

POST ▼ http://localhost:8081/oauth/token

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

Type Basic Auth ▼

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#) ↗

ⓘ Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environ about [variables](#) ↗

Username

Password  ⚠

Body Cookies Headers (10) Test Results ⌐ Status: 200

Pretty Raw Preview Visualize JSON ▼ ≡

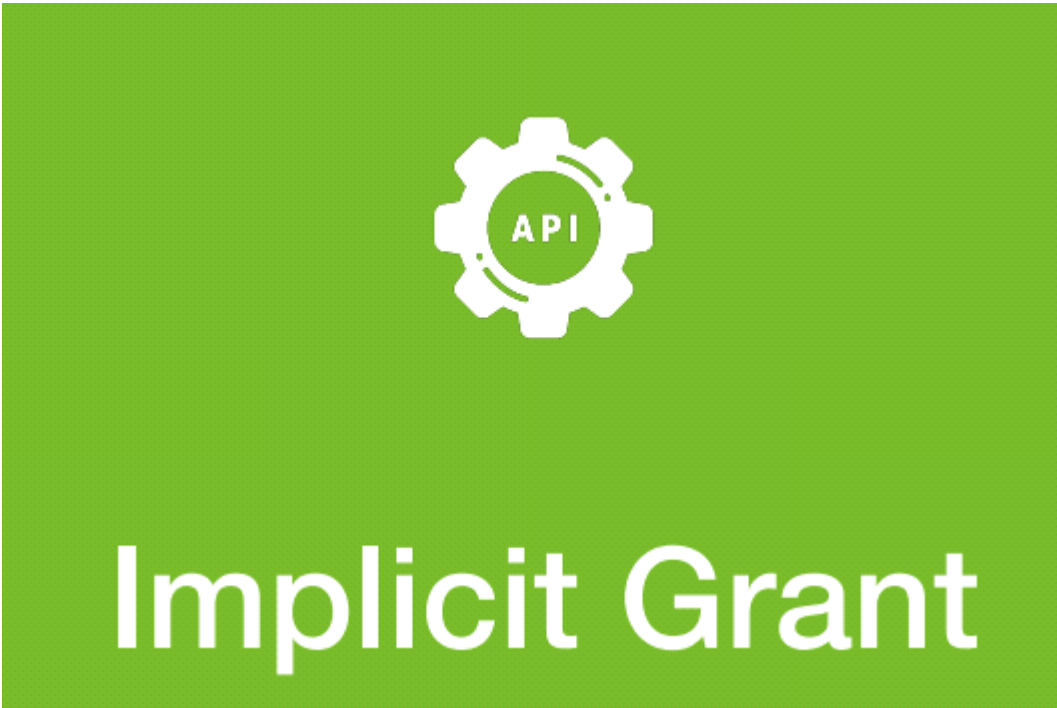


# 22.19. Testando o fluxo Authorization Code com um client JavaScript

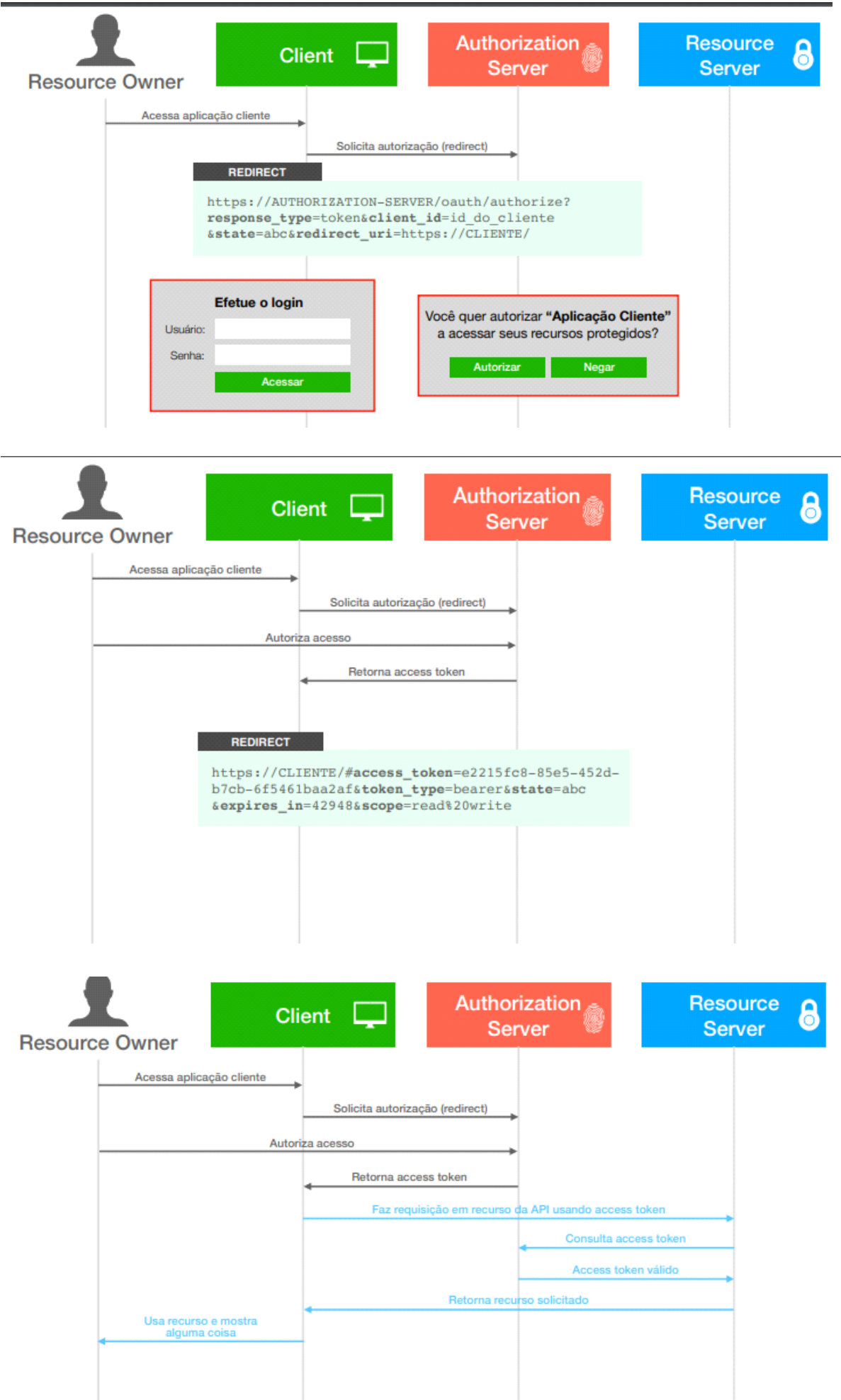
terça-feira, 9 de maio de 2023 14:32

# 22.20. Conhecendo o fluxoImplicit

quinta-feira, 11 de maio de 2023 15:18



Ele é parecido com Authorization Code, porém, mais simplificado. Em vez do Auth Server emitir um code de autorização para o cliente, ele emite diretamente um access token e já retorna na url de callback.



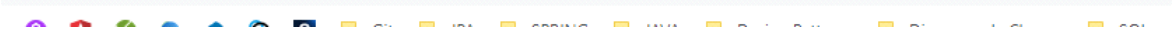
# 22.21. Configurando o fluxo Implicit Grant Type

quinta-feira, 11 de maio de 2023 15:27

```
.and()  
    .withClient( clientId: "webadmin")  
    .authorizedGrantTypes("implicit")  
    .scopes("write", "read")  
    .redirectUri( ...registeredRedirectUri: "http://127.0.0.1:5500")  
;
```

[http://localhost:8081/oauth/authorize?  
response\\_type=token&client\\_id=webadmin&state=abc&redirect\\_uri=http://127.0.0.  
1:5500](http://localhost:8081/oauth/authorize?response_type=token&client_id=webadmin&state=abc&redirect_uri=http://127.0.0.1:5500)

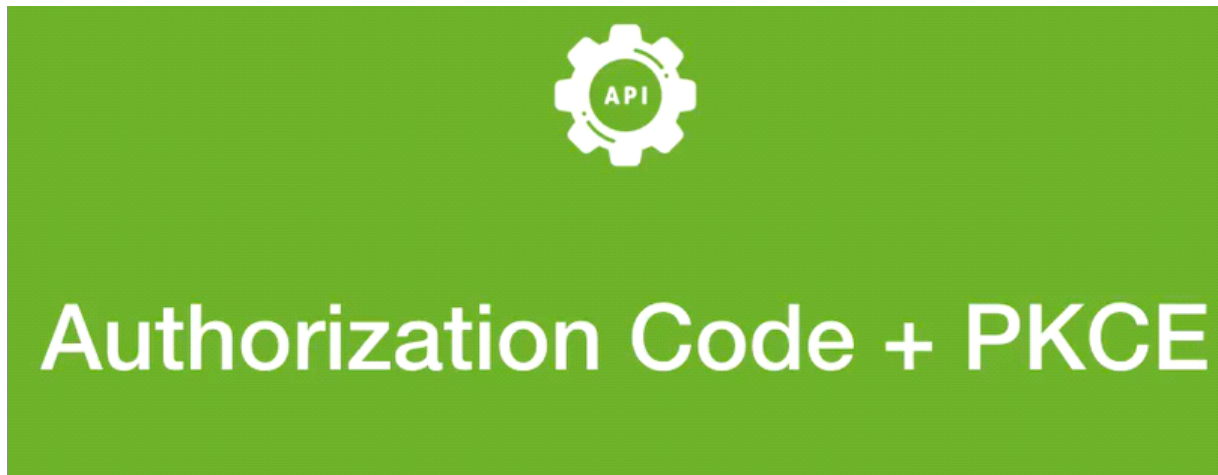
127.0.0.1:5500/#access\_token=N\_xfTnniEBdmIYpTaEQVm8xqgw&token\_type=bearer&state=abc&expires\_in=43199&scope=read%...



## 22.22. Mais segurança com PKCE e Authorization Code Grant

quinta-feira, 11 de maio de 2023

15:41



O oauth2 pode ser estendido por ser uma especificação, logo, podemos estender o OAuth2 com o PKCE, tem como objetivo trazer mais segurança ao Authorization Code.

Problema do Authorization Code:

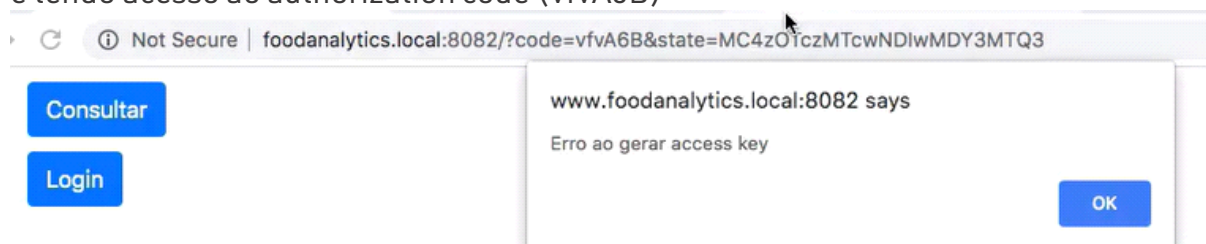
Fluxo sendo usado por um cliente público, pois o cliente público não consegue manter as credenciais (client e secret) confidenciais, pois está visível no código fonte do cliente para o usuário final.

```
const config = {
  clientId: "foodanalytics",
  clientSecret: "food123",
  authorizeUrl: "http://auth.algafood.local:8081/oauth/authorize",
  tokenUrl: "http://auth.algafood.local:8081/oauth/token",
  callbackUrl: "http://www.foodanalytics.local:8082",
  cozinhasUrl: "http://api.algafood.local:8080/v1/cozinhas"
};

let accessToken = "";

function consultar() {
  alert("Consultando recurso com access token " + accessToken);
}
```

e tendo acesso ao authorization code (vfvA6B)



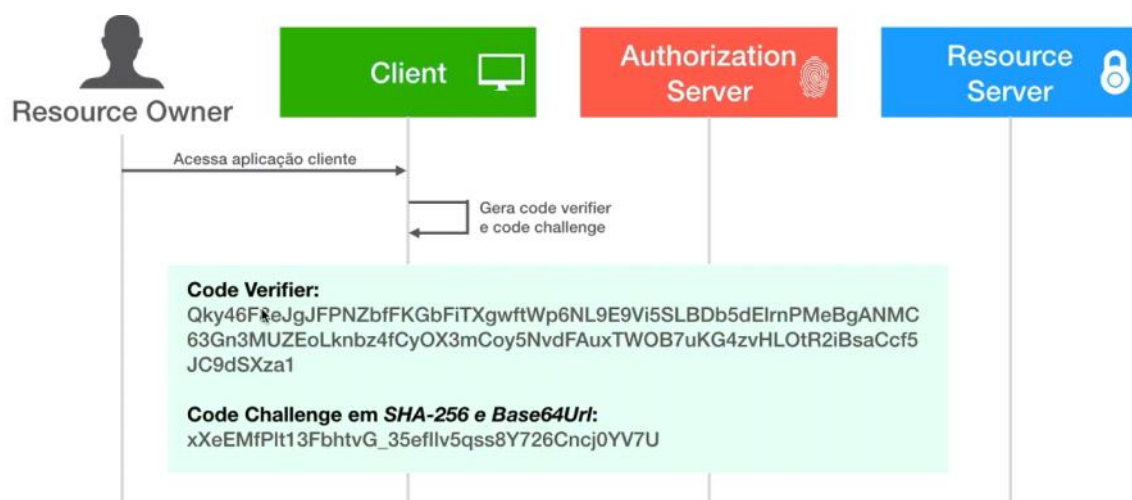
a aplicação está vulnerável para interceptar o code para ter acesso aos recursos protegidos do Resource Server, pois consegue gerar um access token através do code e das credenciais do cliente expostas no frontend.

Um interceptador pode localizar o client id e o client secret em um frontend para acessar os recursos do Authorization Resource.

# Fluxo Authorization Code com PKCE

Mesmo que o código de geração de token possa ser interceptado, o PKCE adiciona uma camada para proteger a camada.

Quando o Resource Owner acessa o Client, antes do Client ser redirecionado para o Authorization Server, o Client gera um code verifier e code challenge.

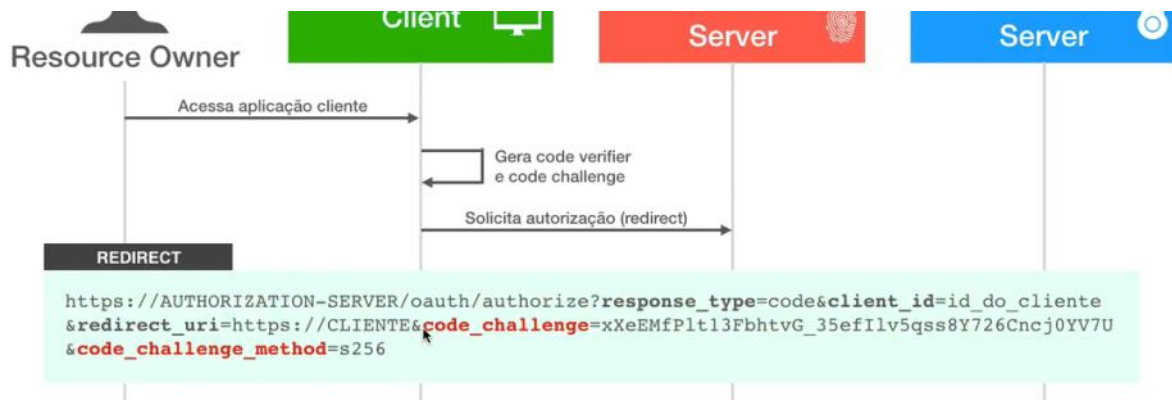


Code Verifier:

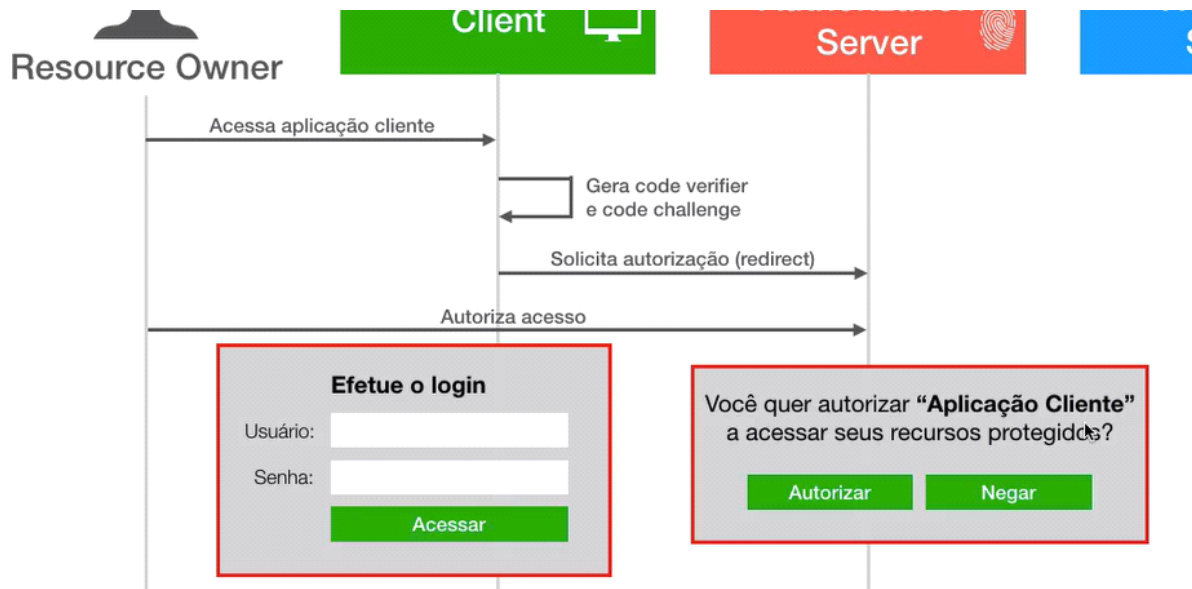
String grande de no mínimo 43 caracteres, e no máximo 128 caracteres, o client gera.

Code Challenge:

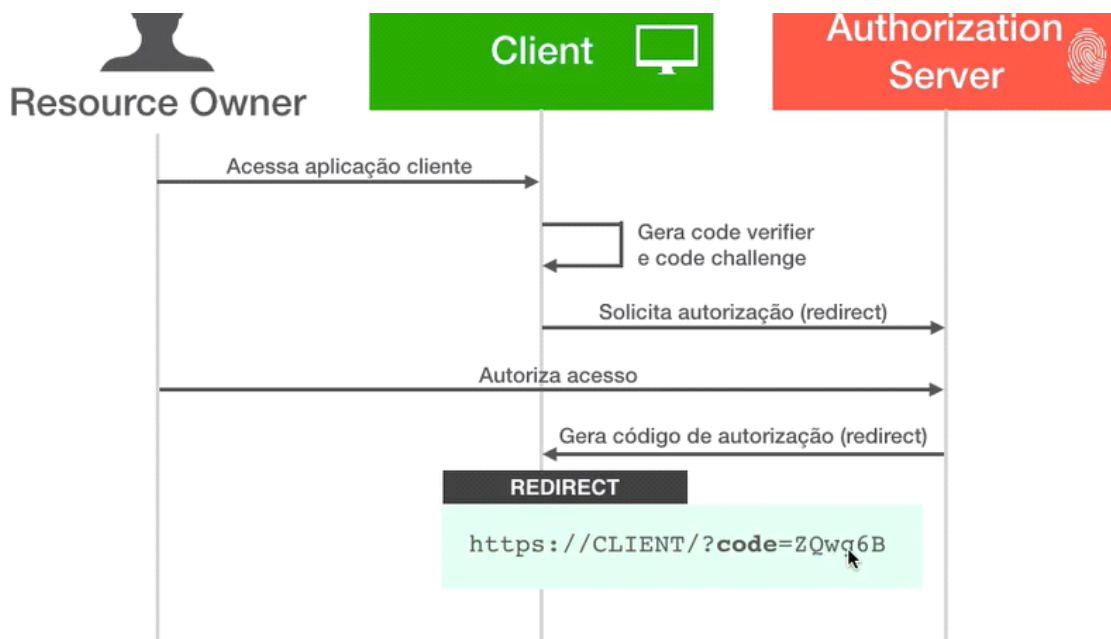
String do code verifier passado por um sha-256 e o resultado disso para pelo base64url.

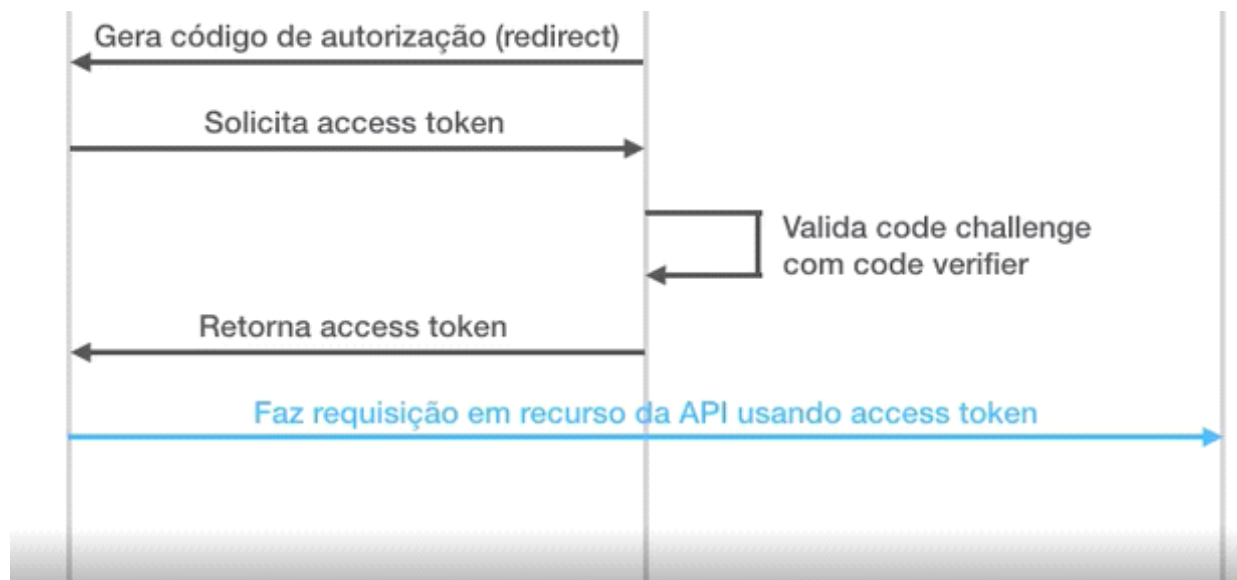
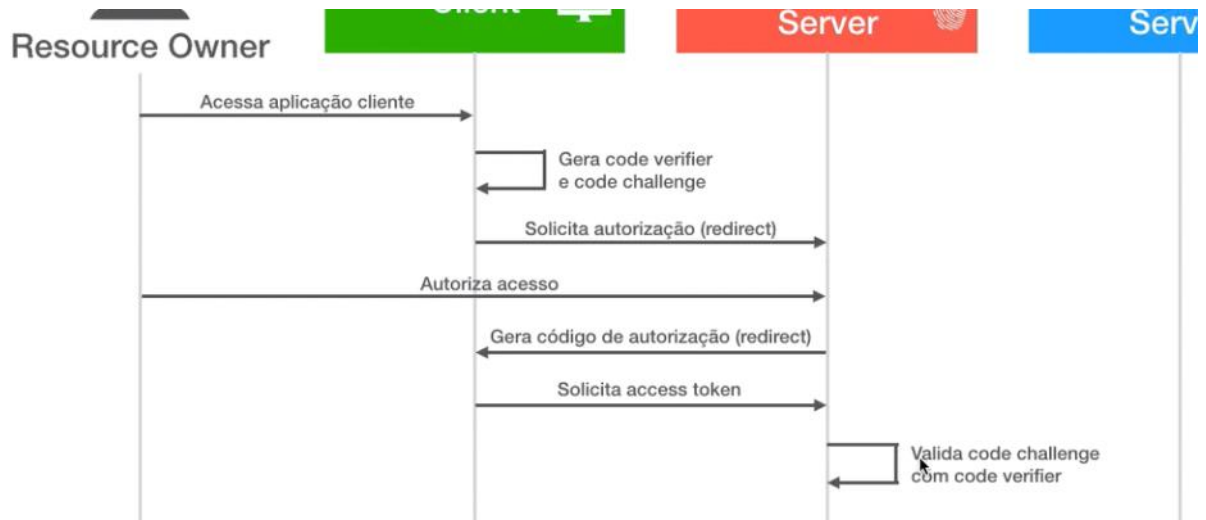
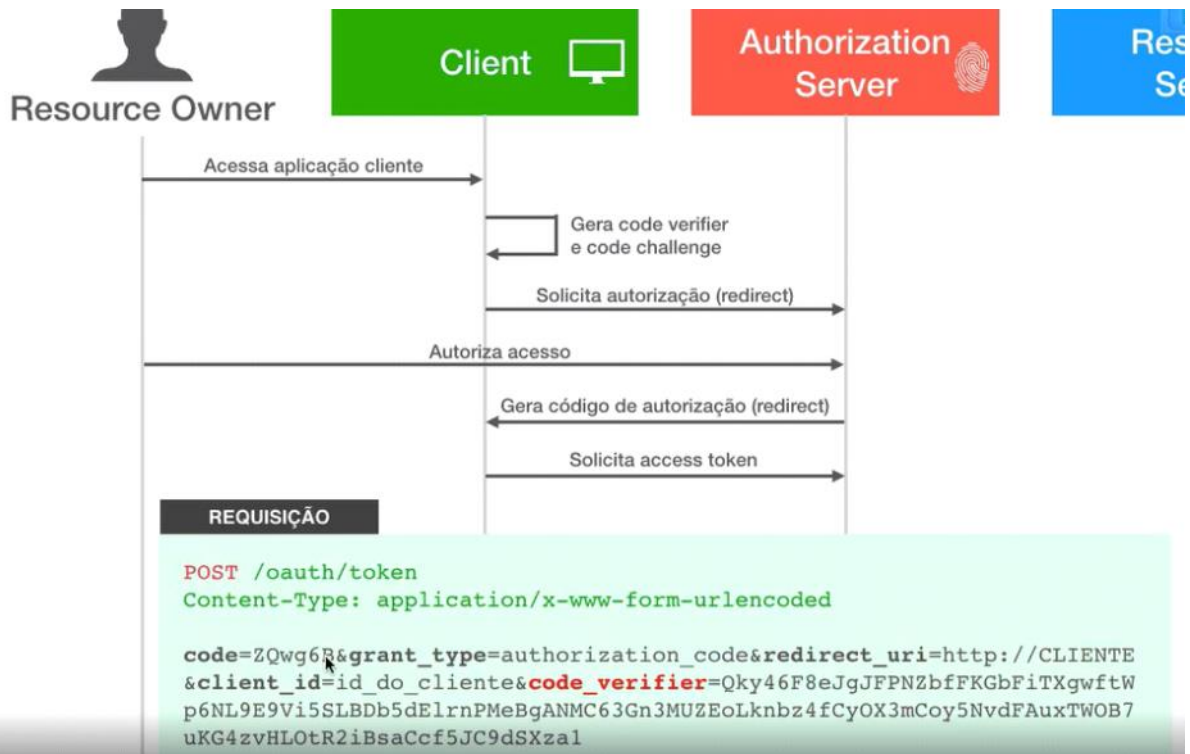


o próprio client gera o code challenge que é passado na url para o resource server



o client terá que ser autorizado pelo server normalmente







## Code Challenge Method: s256

**Code Verifier:**

Qky46F8eJgJFPNZbfFKGbFiTXgwftWp6NL9E9Vi5SLBD5dElrnPMeBgANMC  
63Gn3MUZEoLknbz4fCyOX3mCoy5NvdFAuxTWOB7uKG4zvHLOtR2iBsaCcf5  
JC9dSXza1

**Code Challenge:**

xXeEMfPlt13FbhtvG\_35efllv5qss8Y726Cncj0YV7U

## Code Challenge Method: plain

**Code Verifier:**

Qky46F8eJgJFPNZbfFKGbFiTXgwftWp6NL9E9Vi5SLBD5dElrnPMeBgANMC  
63Gn3MUZEoLknbz4fCyOX3mCoy5NvdFAuxTWOB7uKG4zvHLOtR2iBsaCcf5  
JC9dSXza1

**Code Challenge:**

Qky46F8eJgJFPNZbfFKGbFiTXgwftWp6NL9E9Vi5SLBD5dElrnPMeBgANMC  
63Gn3MUZEoLknbz4fCyOX3mCoy5NvdFAuxTWOB7uKG4zvHLOtR2iBsaCcf5  
JC9dSXza1



# 22.23. Implementando o suporte a PKCE com o fluxo Authorization Code

sexta-feira, 12 de maio de 2023 18:35

<https://gist.github.com/thiagofa/daca4f4790b5b18fed800b83747127ca>

```
package com.algaworks.algafood.auth;

// Solução baseada em: https://github.com/spring-projects/spring-security-oauth/pull/

import ...

public class PkceAuthorizationCodeTokenGranter extends AuthorizationCodeTokenGranter {

    public PkceAuthorizationCodeTokenGranter(AuthorizationServerTokenServices tokenSe
        AuthorizationCodeServices authorizationC
        OAuth2RequestFactory requestFactory) {

        super(tokenServices, authorizationCodeServices, clientDetailsService, request

    }

    1 usage
    @Override
    protected OAuth2Authentication getOAuth2Authentication(ClientDetails client, Token
        OAuth2Authentication authentication = super.getOAuth2Authentication(client, t
```

precisamos habilitar o Token Grant. Adicionar o método no AuthorizationServerConfig

```
81 @ private TokenGranter tokenGranter(AuthorizationServerEndpointsConfigurer endpoints) {
82     var pkceAuthorizationCodeTokenGranter = new PkceAuthorizationCodeTokenGranter(endpoints.getTokenServices(),
83         endpoints.getAuthorizationCodeServices(), endpoints.getClientDetailsService(),
84         endpoints.getOAuth2RequestFactory());
85
86     var granters :List<TokenGranter> = Arrays.asList(
87         pkceAuthorizationCodeTokenGranter, endpoints.getTokenGranter());
88
89     return new CompositeTokenGranter(granters);
90 }
```

O método fornece e a instância de um TokenGranter que contém a configuração de todos os tipos de TokenGranter suportado pelo OAuth2

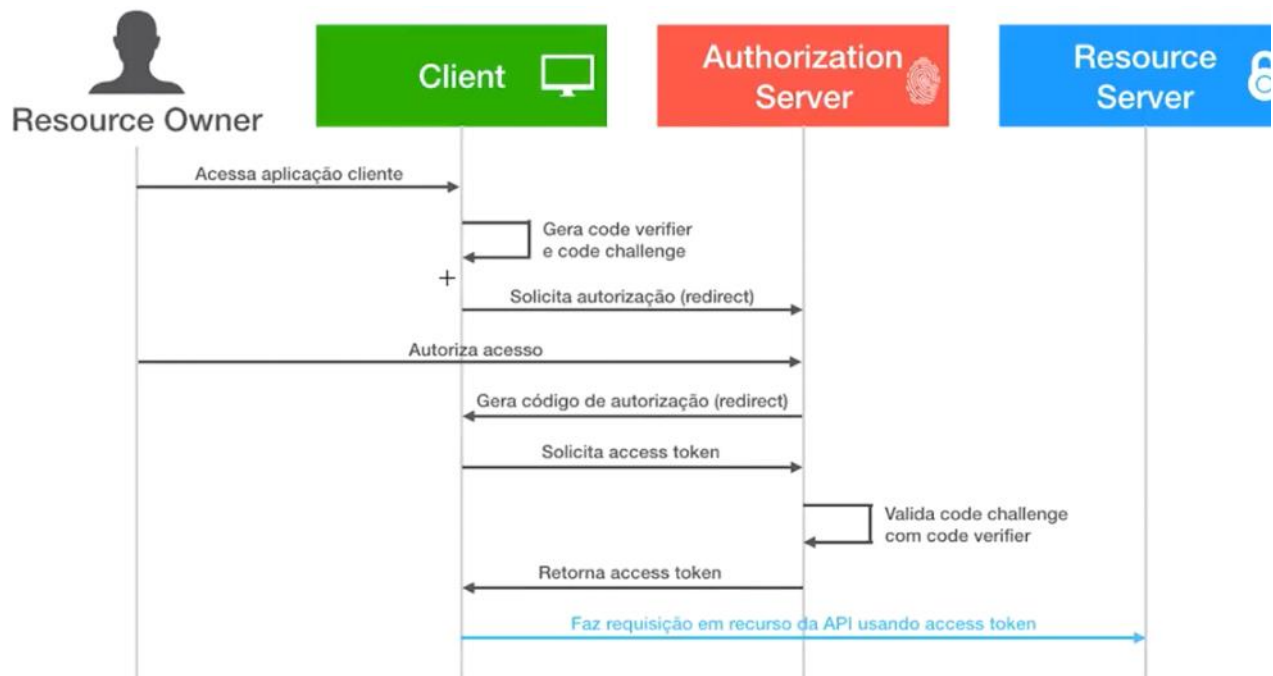
```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints, AuthenticationManager authenticationManager,
    endpoints.userDetailsService(userDetailsService);
    endpoints.reuseRefreshTokens(false);
    endpoints.tokenGranter(tokenGranter(endpoints));
}
```

com isso todos os tipos de TokenGranter estão sendo chamados e a aplicação rodará normalmente. Não faz quebrar.

## 22.24. Testando o fluxo Authorization Code com PKCE com o método plain

sexta-feira, 12 de maio de 2023

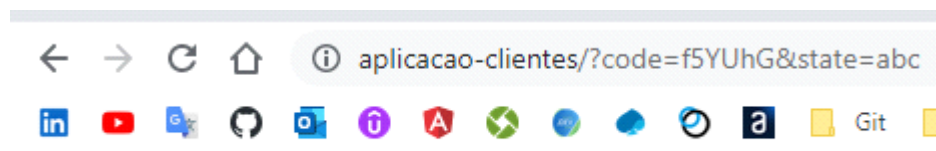
20:42



[http://localhost:8081/oauth/authorize?response\\_type=code&client\\_id=foodanalytcs&redirect\\_uri=http://aplicacao-clientes&code\\_challenge=&code\\_challenge\\_method=plain](http://localhost:8081/oauth/authorize?response_type=code&client_id=foodanalytcs&redirect_uri=http://aplicacao-clientes&code_challenge=&code_challenge_method=plain)

Code Verifier= teste123

Code Challenge= no método plain é o mesmo Code Verifier



código de acesso gerado

POST ▼ http://localhost:8081/oauth/token

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

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

Key	Value
<input checked="" type="checkbox"/> grant_type	authorization_code
<input checked="" type="checkbox"/> code	f5YUhG
<input checked="" type="checkbox"/> redirect_uri	http://aplicacao-clientes
Key	Value

Ao utilizar o Pkce, precisamos de um code verifier gerado no cliente

Body Cookies Headers (12) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```

1 {
2   "error": "invalid_grant",
3   "error_description": "Code verifier expected."
4 }
```

[http://localhost:8081/oauth/authorize?response\\_type=code&client\\_id=foodanalytcs&redirect\\_uri=http://aplicacao-clientes&code\\_challenge=abcdef123456&code\\_challenge\\_method=plain](http://localhost:8081/oauth/authorize?response_type=code&client_id=foodanalytcs&redirect_uri=http://aplicacao-clientes&code_challenge=abcdef123456&code_challenge_method=plain)

até o momento, estamos utilizando a autenticação do cliente pelo basic auth

AlgaFood - Authorization Server / Authorization Code - Access Token Copy

POST ▼ http://localhost:8081/oauth/token

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

Type Basic Auth ▼

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#) ↗

ⓘ Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, please use the [variables](#) feature.

Username

Password  ⚠

porém, ao utilizar o Pkce, podemos omitir a autenticação do cliente pelo Basic Auth e utilizar forms. Temos que configurar o Auth Server adicionando uma senha vazia e poderíamos autenticar o cliente enviando uma senha vazia no campo "Password", mas podemos tirar a autenticação do client e no body utilizando um form.

```

.and()
.withClient( clientId: "foodanalitycs")
.secret(passwordEncoder.encode( rawPassword: "I"))
.authorizedGrantTypes("authorization_code")
.scopes("write", "read")

```

```

/*Para configurar o acesso ao endpoint de checagem de token ou check token, introspecção de token */
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    //security.checkTokenAccess("isAuthenticated()");//Spring Security Expression.
    security.checkTokenAccess("permitAll()").allowFormAuthenticationForClients(); //permitindo qualquer cliente
    // P/ acessar o check token, precisa de autenticação do cliente
}

```

Params
Authorization
Headers (8)
Body ●
Pre-request Script
Tests
Settings

● none
● form-data
● x-www-form-urlencoded
● raw
● binary
● GraphQL

	Key	Value
<input checked="" type="checkbox"/>	grant_type	authorization_code
<input checked="" type="checkbox"/>	code	IOJFa_
<input checked="" type="checkbox"/>	redirect_uri	http://aplicacao-clientes
<input checked="" type="checkbox"/>	code_verifier	abcdef123456
<input checked="" type="checkbox"/>	client_id	foodanalitycs
<input type="checkbox"/>	client_secret	
	Key	Value

também poderíamos adicionar um client\_secret

POST

http://localhost:8081/oauth/token

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	Key	Value
<input checked="" type="checkbox"/>	grant_type	authorization_code
<input checked="" type="checkbox"/>	code	rfTdcy
<input checked="" type="checkbox"/>	redirect_uri	http://aplicacao-clientes
<input checked="" type="checkbox"/>	code_verifier	abcdef123456
<input checked="" type="checkbox"/>	client_id	foodanalytics
<input type="checkbox"/>	client_secret	

	Key	Value
--	-----	-------

BodyCookiesHeaders (13)Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "access_token": "jJfoH-q5koNEZUcbZIVn2AJQZUA",
3   "token_type": "bearer",
4   "expires_in": 43199,
5   "refresh_token": "..."
6 }
```

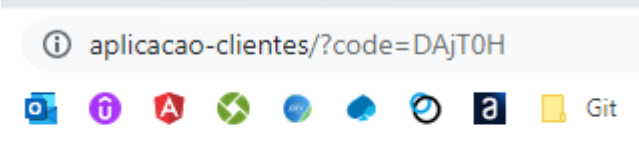
# 22.25. Testando o fluxo Authorization Code com PKCE com o método SHA-256

sexta-feira, 12 de maio de 2023 22:03

[http://localhost:8081/oauth/authorize?response\\_type=code&client\\_id=foodanalitycs&redirect\\_uri=http://aplicacao-clientes&code\\_challenge=mruCOFzjul34iD7LvZX0paBHM8a1PTvbS2WHdgXyxGQ&code\\_challenge\\_method=s256](http://localhost:8081/oauth/authorize?response_type=code&client_id=foodanalitycs&redirect_uri=http://aplicacao-clientes&code_challenge=mruCOFzjul34iD7LvZX0paBHM8a1PTvbS2WHdgXyxGQ&code_challenge_method=s256)

o cliente tem que ter uma função para gerar o code verifier aleatoriamente entre 46 e 128 caracteres.

o client gera o code verifier e também o code challenge, o code challenge é adicionado na url para o Auth Server, o Auth Server gera o código de acesso



com o codigo de acesso, fazemos uma requisição para gerar o token

AlgaFood - Authorization Server / Authorization Code - Access Token Copy

POST

http://localhost:8081/oauth/token

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	Key	Value
<input checked="" type="checkbox"/>	grant_type	authorization_code
<input checked="" type="checkbox"/>	code	DAjT0H
<input checked="" type="checkbox"/>	redirect_uri	http://aplicacao-clientes
<input checked="" type="checkbox"/>	code_verifier	iGaiF5-Fjj3WJkSQGCVy8iPHrVFEB4RooajBhWKWQ8
<input checked="" type="checkbox"/>	client_id	foodanalitycs
<input type="checkbox"/>	client_secret	
	Key	Value

BodyCookiesHeaders (13)Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

"access\_token": "jJfoH-q5koNEZUcbZIvn2AJQZUA",

"token\_type": "bearer",

"expires\_in": 42673,

"scope": "read write"

tonyxu-io.github.io/pkce-generator/

Git

JPA

SPRING

JAVA

Design Patterns

Dis

Online PKCE Generator Tool

An online tool to generate code verifier and code challenge for OAuth with PKCE.

### Code Verifier

This tool serves as an example implementation or for sending manual requests. Never reuse code ve

iGaiF5-Fjj3WJkSQGCVy8iPHrVFEB4RooajBhWKWQ8

### Code Challenge

mruCOFzjul34iD7LvZX0paBHM8a1PTvbS2WHdgXyxGQ

Generate Code Challenge

Generate Code Verifier

# 22.27. Decidindo qual fluxo OAuth2 usar

quinta-feira, 11 de maio de 2023 16:06

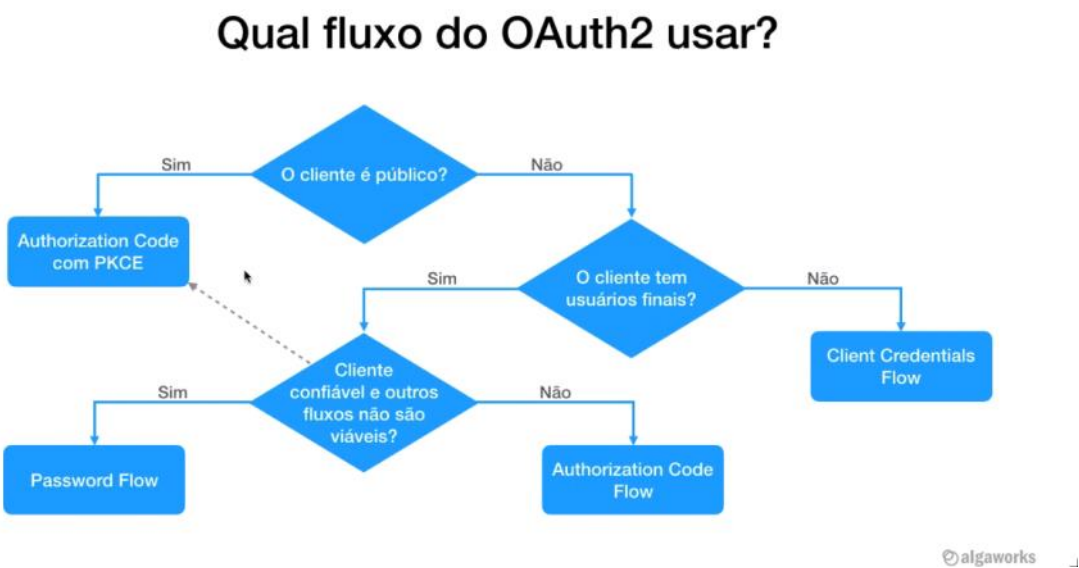
## Tipo de cliente

- Público
  - Quando o usuário da aplicação frontend, de alguma maneira, conseguir acessar o clientId e clientSecret, como aplicação javascript que pode ser acessando usando a ferramenta de dev ne um navegador, visualizando o código fonte
- Confidencial.
  - Quando o código fonte não pode ser acessado pelo usuário, como o frontend vem do servidor

## Cliente confiável vs cliente não confiável

Cliente confiável:  
aplicação própria, da empresa, ou do time de desenv.

Cliente não confiável:  
aplicação terceira, que usa o Auth Server, mas não está sob o seu poder, os reais objetivos são desconhecidos.



# Credenciais automáticas no Postman

sábado, 6 de maio de 2023 00:22

ET Root Entry PointGET cozinhas - listarPOST Password Flow

AlgaFood / Cozinha / cozinhas - listar

GET

http://localhost:8080/v1/cozinhas?size=1&page=0&sort=

ParamsAuthorizationHeaders (8)BodyPre-request Script

Type

The authorization data will be automatically added to the request and sent the request. Learn more about [OAuth 2.0](#).

Add authorization data to

OAuth 2.0

Inherit auth from parent

No Auth

API Key

Bearer Token

JWT Bearer

Basic Auth

Digest Auth

OAuth 1.0

OAuth 2.0

Hawk Authentication

AWS Signature

NTLM Authentication

Akamai EdgeGrid

Header F

Current Token

This token

Token

Header F

Auto-refresh

Your expires in

Share to

This will

Response

AuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

Share token

This will allow anyone with access to this request to view and use it.

Configure New Token

Token Name

Token AlgaFood

Grant Type

Password Credentials

Access Token URL

http://localhost:8081/oauth/token

Client ID

algafood-web

Client Secret

web123

Username

Gustavo

Password

...

Scope

e.g. read:org

Client Authentication

Send as Basic Auth header

> Advanced

Clear cookies

Get New Access Token



# Utilizar Token no Postman

sábado, 6 de maio de 2023 00:30

GET

http://localhost:8080/v1/cozinhas?size=1&page=0&sort=nome

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Type

Bearer Token

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, Postman has automatically redacted some values. [Learn more](#)

Token

-lpr51ODrNdeoLC9TSrqIOEC3\_Q