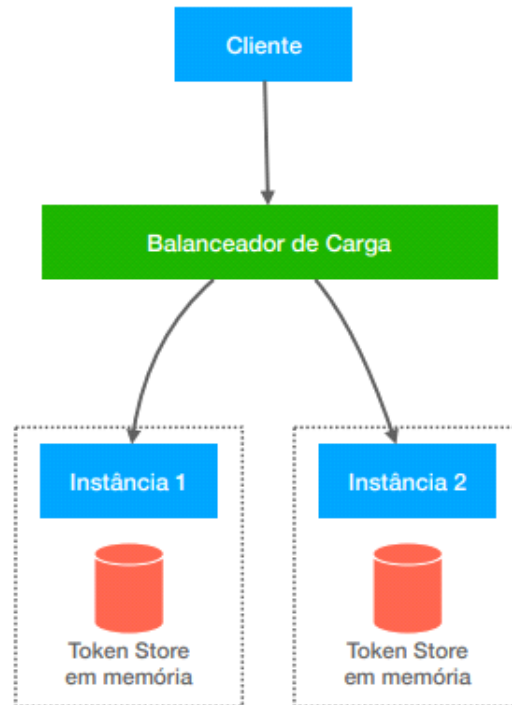


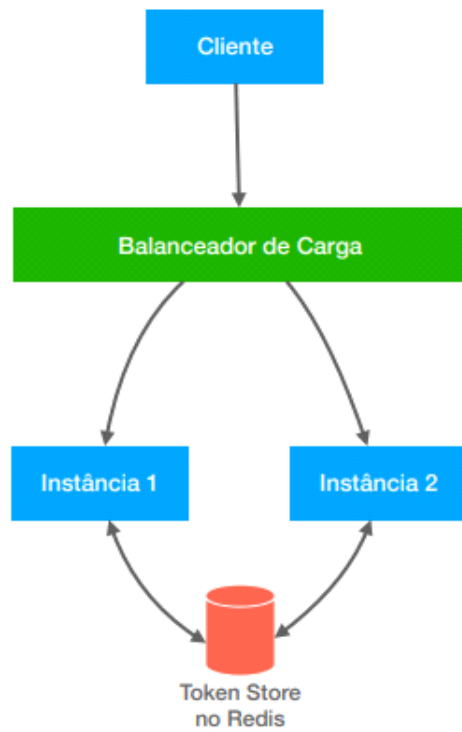
23.1. Armazenando tokens no Redis: um banco de dados NoSQL

quinta-feira, 11 de maio de 2023 16:43

Ao reiniciar uma aplicação, tanto o token de acesso quanto o refresh token serão inutilizados, mesmo estando em um período de tempo válido. Ele não está sendo armazenado em um banco de dados ou qualquer outro lugar que tenha uma persistência maior. Está sendo armazenado em memória.



Ao fazer o deploy da aplicação a nuvem, geralmente temos mais de uma instância da aplicação rodando, para termos o balanceamento de carga. Logo, o token gerado por um usuário pode estar rodando em memória de uma instância e ser autenticado em outra instância do servidor. Temos que armazenar os tokens em um local, isso se chama token store.



Podemos utilizar o Redis (No-Sql) para armazenamento de tokens do Authorization Server

23.2. Configurando o RedisTokenStore

sexta-feira, 12 de maio de 2023

23:14

Armazenamento de tokens no Redis

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
</dependencies>
```

```
C:\Windows\System32\cmd.exe - memurai-cli.exe

Microsoft Windows [versão 10.0.19045.2965]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Program Files\Memurai>memurai-cli.exe
127.0.0.1:6379> pinh
(error) ERR unknown command 'pinh', with args beginning with:
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> redis-server
(error) ERR unknown command 'redis-server', with args beginning with:
127.0.0.1:6379> keys *
1) "refresh_auth:CnbuuI9VjHG-qgFubxWxwez66JM"
2) "auth_to_access:ae8d8ce93d40d0a21dd3680765b3dd58"
3) "auth:kZK9akbE1z_qjEtKNoGHE-KbTMg"
4) "access:kZK9akbE1z_qjEtKNoGHE-KbTMg"
5) "uname_to_access:algafood-web:Gustavo"
6) "refresh:CnbuuI9VjHG-qgFubxWxwez66JM"
7) "refresh_to_access:CnbuuI9VjHG-qgFubxWxwez66JM"
8) "access_to_refresh:kZK9akbE1z_qjEtKNoGHE-KbTMg"
9) "client_id_to_access:algafood-web"
127.0.0.1:6379> _
```

configurar no application.properties do Authorization Server

```
#23.2. Configurando o RedisTokenStore
spring.redis.host=localhost
spring.redis.password=
spring.redis.port=6379
```

Em Authorization Server Config

```
1 usage
public TokenStore tokenStore() {
    return new RedisTokenStore(redisConnectionFactory);
}
```

```

@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints.authenticationManager(authenticationManager); //somente o fluxo "password" pr
    endpoints.userDetailsService(userDetailsService); /*para refresh_token*/
    endpoints.reuseRefreshTokens(false);
    endpoints.tokenGranter(tokenGranter(endpoints));
    endpoints.tokenStore(tokenStore());
}

```

```

1 usage
@Autowired
private RedisConnectionFactory redisConnectionFactory;

```

Agora, mesmo que a aplicação no servidor reinicie, os tokens serão armazenados em memória.

```

127.0.0.1:6379> redis-server
(error) ERR unknown command 'redis-server', with args
127.0.0.1:6379> keys *
1) "refresh_auth:CnbuuI9VjHG-qgFubxWxwez66JM"
2) "auth_to_access:ae8d8ce93d40d0a21dd3680765b3dd58"
3) "auth:kZK9akbE1z_qjEtKNoGHE-KbTMg"
4) "access:kZK9akbE1z_qjEtKNoGHE-KbTMg"
5) "uname_to_access:algafood-web:Gustavo"
6) "refresh:CnbuuI9VjHG-qgFubxWxwez66JM"
7) "refresh_to_access:CnbuuI9VjHG-qgFubxWxwez66JM"
8) "access_to_refresh:kZK9akbE1z_qjEtKNoGHE-KbTMg"
9) "client_id_to_access:algafood-web"
127.0.0.1:6379> keys a*
1) "auth_to_access:ae8d8ce93d40d0a21dd3680765b3dd58"
2) "auth:kZK9akbE1z_qjEtKNoGHE-KbTMg"
3) "access:kZK9akbE1z_qjEtKNoGHE-KbTMg"
4) "access_to_refresh:kZK9akbE1z_qjEtKNoGHE-KbTMg"
127.0.0.1:6379>

```

limpar todos os registros no redis

```

4) "access_to_refresh:kZK9akbE1z_
127.0.0.1:6379> flushall
OK
127.0.0.1:6379>

```

23.3. Entendendo a diferença entre Stateful e Stateless Authentication

sábado, 13 de maio de 2023 08:45



Stateful vs Stateless Authentication

Stateful vs Stateless

Stateful: com estado

Stateless: sem estado

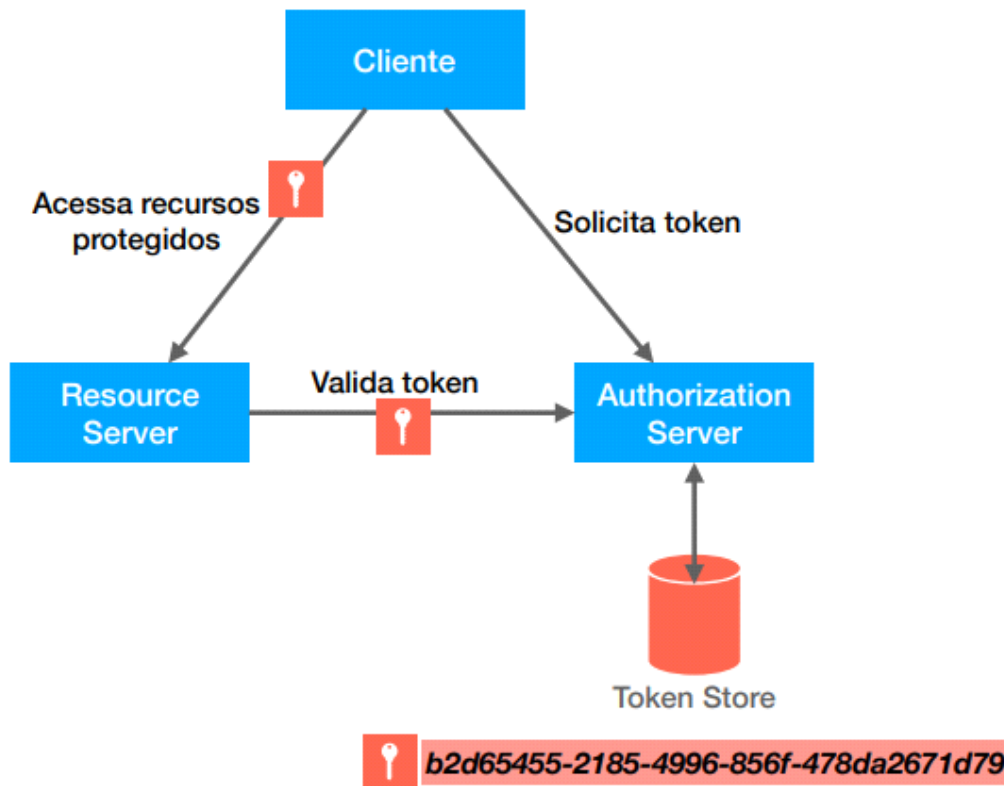
Uma das constraints para uma api ser Restful é que "A comunicação deve ser stateless (sem estado) por natureza, de forma que cada requisição do cliente para o servidor deve conter todas as informações necessárias para entendê-la e não pode usar informações contextuais armazenadas no servidor. O estado da sessão deve ser armazenado inteiramente no cliente." - Roy Fielding

Application State vs Resource State

O estado da aplicação que deve ser stateless, pois difere do armazenamento de informações dos recursos. O estado da aplicação é o mesmo estado da sessão.

Em uma aplicação e-commerce por exemplo, ao adicionar produtos em um carrinho, o cliente faz várias requisições para ter acesso às informações de um produto específico para ir montando o carrinho de compras, são requisições intermediárias para completar uma ação, isso é stateful (no lado do cliente) porque armazena informações contextuais, que não faz parte da real necessidade, que é realizar a compra, mas que faz parte para alcançá-la. Trazendo isso para nossa api, é como se, em uma requisição, o servidor irá enviar todas as informações necessárias para compor o carrinho de compras ou qualquer tipo de informação que o cliente necessitar, e nunca vai armazenar as informações parcialmente daquela requisição para, posteriormente, realizar outra e enviar o restante das informações. Ao logo do curso foi abordado a Stateful.

Stateful Authentication



Até agora estamos utilizando um Stateful Authentication por conta do uso do Opaque Token, o código do token não significa nada, é apenas uma String aleatória, o Cliente utiliza o token para acessar os recursos protegidos do Resource Server e o Resource Server precisa validar o token para todas as requisições pois, como o Token é opaco, ele não possui informações extras do cliente.

Vantagens:

- É capaz de revogar/invalidar
- Dados relacionados ao token podem ser alterados

Desvantagens:

- Infraestrutura do lado do Auth servidor
- Cria uma dependência entre Resources e Auth Server, podemos ter vários Resources Server e um único Auth Server, e com isso uma forte dependência
- O Auth Server vira um single point of failure, porém, pode ser minimizado com novas instâncias do Auth Server

Stateless Authentication

Utilizado para suprir as desvantagens do Stateful. Os dados da sessão são armazenados no lado do cliente, em uma aplicação web, são armazenados no Browser.

Transparent Token

Contrário de Token Opaco, ele é uma String baseada em um JSON da sessão do usuário, com informações do próprio usuário e tempo de expiração, depois passa por uma codificação Base64url para transformar em uma String (não é criptografia) e pode ser decodificado.

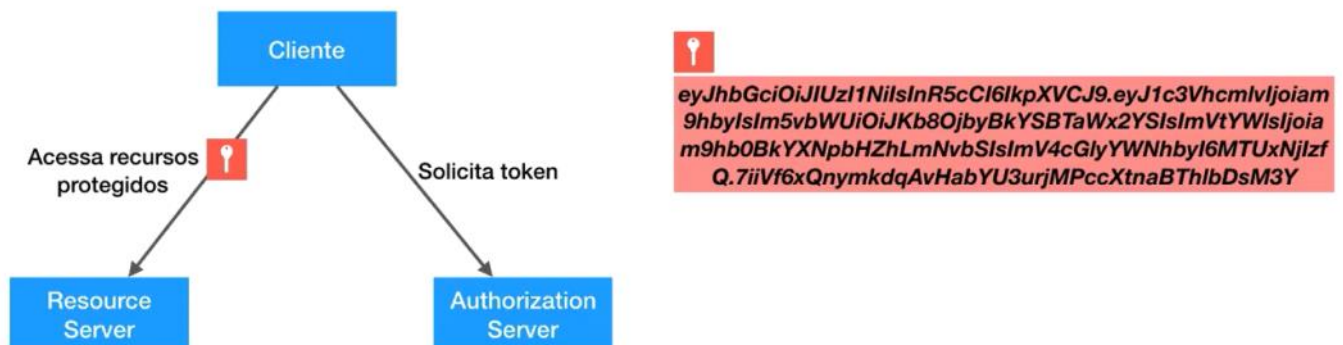
```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3VhcmVlIjoiam9hbyIsIm5vbWUiOiJKb8OjbyBkYSBTaWx2YSIsImVtYWlsIjoia m9hb0BkYXNpbHZhLmNvbSIsImV4cGlyYW5hbyI6MTUxNjlf Q.7iiVf6xQnymkdqAvHabYU3urjMPccXtnaBThlbDsM3Y
```

É um token transparente assinado e com o JSON autocontido abaixo

```
{
  "usuario": "joao",
  "nome": "João da Silva",
  "email": "joao@dasilva.com",
  "expiracao": 151623
}
```

Além disso, também possui um hash para verificar integridade.

Stateless Authentication



O cliente passa um JSON para o Auth Server e o Auth Server devolve um token transparente para o Cliente acessar os Recursos Protegidos. O Authorization Server não deve armazenar esse token (por ser uma aplicação Stateful) porque contém informações do usuário.

O cliente acessa recursos protegidos do RS, o mesmo verifica a integridade do token através da assinatura contida nele, e não precisa do AS para validar o Token.

Vantagens:

Não necessita de infraestrutura no lado do servidor

Como o RS não precisa do AS para verificar integridade do Token, eles não possuem dependência para funcionar

Desvantagens:

- Não revogação de token

- Mais dados trafegados na requisição, quanto maior o JSON maior o Token gerado

- Uma vez emitido o token com os dados, o usuário não poderá alterar as suas informações para não ter inconsistência com as informações antigas dentro do Token (não é possível revogar)

Qual solução escolher?

Foco em resolver a solução da demanda

23.4. Transparent Tokens: conhecendo o JSON Web Tokens (JWT)

sábado, 13 de maio de 2023 13:55

O JWT é um tipo de token transparente, e um padrão de transporte de dados no formato json de maneira compacta especificada na RFC 7519 <https://datatracker.ietf.org/doc/html/rfc7519>.

- String codificada em Base64 e assinado com um hash para assinatura
- Contém 3 parte divididos por um ponto. Header, Payload e Assinatura

Header: informações sobre o token, como o tipo e o algoritmo de codificação

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NzgwMzExNTcsInVzZXJfbmFtZSI6InRoaWFnb3R5b3R5ImF1dG8vcm10aWVzIjpbIlJPTEVfQURNSU4iXSswianRpIjo1YzIxMjBkMjYtOGY1My00OD1lLTkyMjMtNTQwOTBiYTE0NjRmIiwiaWY2xpZW50X21kIjo1YWxnYWZvb2Qtd2ViIiwic2NvcGUiOi01sid3JpdGUiLCJyZWFKI119.16-gyVamsfrewLrHB2v_pVGhbUrvPhQqbo96P-oE7rE
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "exp": 1578831157,  "user_name": "thiago",  "authorities": [    "ROLE_ADMIN"  ],  "iat": 1578831157}
```

Payload contém clains: afirmações/propriedades que contém informações sobre o que o token representa em um par de chave-valor. Cada chave-valor é uma clains. Existe convenções de nomeações de algumas clains como "exp" de expiration time (data/hora de expiração), "jti" identificador único ou clains customizadas.

Não deve ser armazenada informações sensíveis como senhas de usuários.

Assinatura: hash gerado usando o algoritmo definido no Header

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret  ) ☐ secret base64 encoded
```

O hash gerado precisa de um "secret" para servir de assinatura

Utilizado para autenticação Stateless

23.5. Gerando JWT com chave simétrica (HMAC SHA-256) no Authorization Server

sábado, 13 de maio de 2023 14:19

Configurar o Authorizatin Server para emitir JWT com assinatura usando o algoritmo HMACSHA-256, que é um criptografia simétrica, ou seja, uma única senha/secret que é compartilhada para quem emite o JWT e quem recebe também no client ou consumidor.

Quem emite no caso é o A.S e depois o R.S precisa ter acesso ao secret para verificar a integridade do token. Quem tentar burlar o acesso ao R.S com um JWT criado e que não tenha o secret, vai ter uma falha.

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-jwt</artifactId>
  <version>1.1.1.RELEASE</version>
</dependency>
```

no AuthorizationServerConfig

```
1 usage
@Bean/*Converte informações de user Logado para JWT * pode ser usado como Bean*/
public JwtAccessTokenConverter jwtAccessTokenConverter () {
    /*utiliza hmacsha-256 simétrico*/
    JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
    jwtAccessTokenConverter.setSigningKey("algaworks");

    return jwtAccessTokenConverter;
}
```

```
configurar accessTokenConverter()
```

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    endpoints.authenticationManager(authenticationManager). //somente o fluxo "password" pr
    userDetailsService(userDetailsService)./*para refresh_token*/
    reuseRefreshTokens(false).
    tokenGranter(tokenGranter(endpoints)).
    accessTokenConverter(jwtAccessTokenConverter());
}
```

AlgaFood - Authorization Server / Password Flow - 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	Description
<input checked="" type="checkbox"/> username	Gustavo	
<input checked="" type="checkbox"/> password	123	
<input checked="" type="checkbox"/> grant_type	password	

Key	Value	Description
-----	-------	-------------

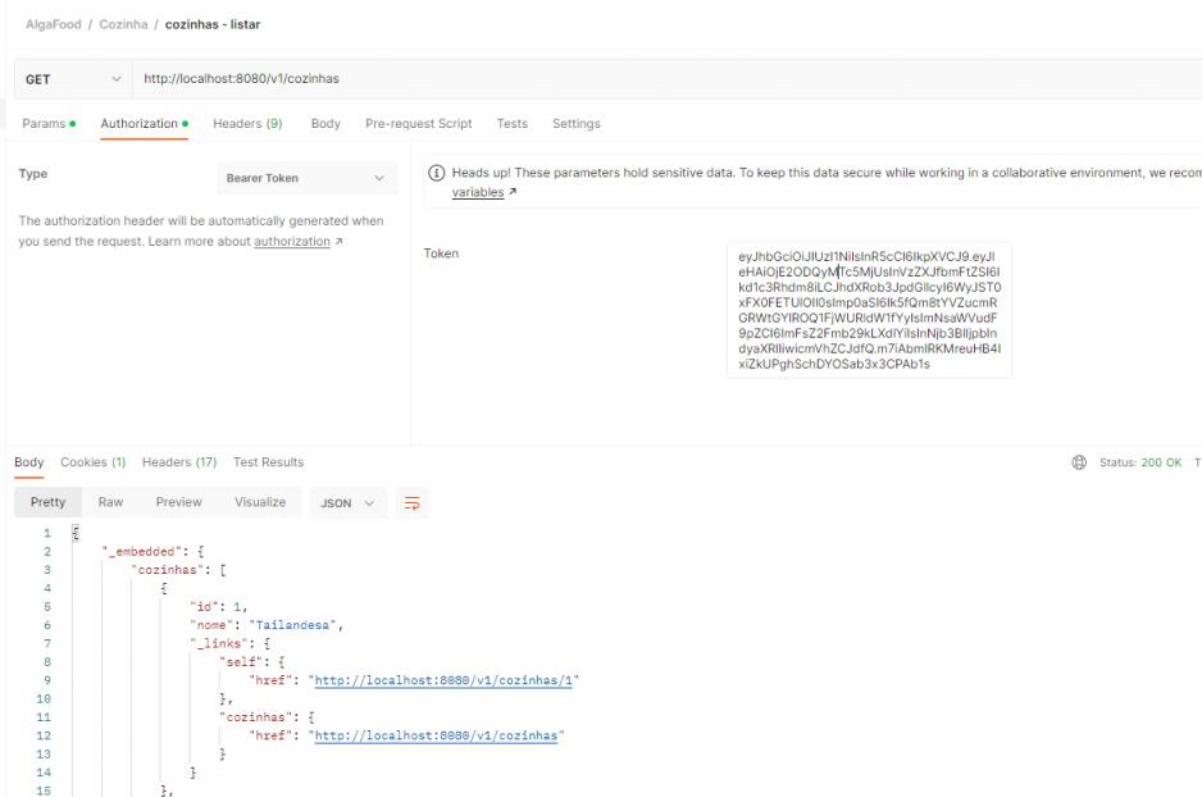
```
Body Cookies Headers (13) Test Results
Pretty Raw Preview Visualize JSON ↕
1
2 "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjEwMjQyMTc5OTY5InVzZXJibmFtZSI6IkdldkI3Rhbm8lICJhdXRob3JpdGlcyI6WyJST0xFeX0FETU0l1l0Imp8aSI6Ik5fQm8tYVZucmRGWtGYlROQ1FjWURpbIndyaXRlIiwicmVhZCJdfQ.m7iAbmIRKMrzuHB4IxizKUphgSchdYOVSab3x3CPAb1s",
3
4 "token_type": "bearer",
5
6 "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VyZWZhbnBhbmU0IjHdXN0eXZvIiwic2NvcGU0I0lscid3JpdGUlICJyZWFiKSIlImF0aSI6Ik5fQm8tYVZucmRGWtGYlROQ1FjWURLldWifYysImV4Ci6MTY04OTE4NTktSTWtIOEkzs19jSTlSc2JxZV9aOGt3TKkiLCJjbGllbnRfaWQiOiJhbGdhZm9vZC13ZWliffQ.MMZSSdIIoDPuDhg1HI6ys97EPzMDg20kjvMzH1VyTg",
7
8 "expires_in": 215999,
9
10 "scope": "write read",
11
12 "jti": "N_Bo-aVnzdFEkFBTNCQCYDeum_c"
```

agora o token não é invalidado ao reiniciar o servidor.

é possível também acessar recursos protegidos, porém, o R.S é dependente do A.S, temos que modificá-lo para ser completamente s

23.6. Configurando o Resource Server para JWT assinado com chave simétrica

sábado, 13 de maio de 2023 15:24



É possível acessar recursos protegidos mas o R.S ainda é dependente do A.S, pois configuramos o R.S para fazer a verificação de **opaque tokens** no **application.properties**, para que chame o endpoint de checagem.

```
#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=checktoken
spring.security.oauth2.resourceserver.opaquetoken.client-secret=check123
```

Na classe ResourceServerConfig do Resource Server, configuramos para trabalhar cm JWT

```
@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()
            .cors()
            .and()
            .oauth2ResourceServer()
                .opaqueToken();
        .jwt();
}

@Bean
public JwtDecoder jwtDecoder(){
    final SecretKeySpec secretKeySpec = new SecretKeySpec("algaworks".getBytes(), algorithm: "HmacSHA256");
    return NimbusJwtDecoder.withSecretKey(secretKeySpec).build();
}
```

Nota: a chave secreta tem que ter no mínimo 32 caracteres e tem que ser a mesma key configurada no A.S classe AuthServerConfig

```
1 usage
@Bean/*Converte informações de user logado para JWT * pode ser usado como Bean*/
public JwtAccessTokenConverter jwtAccessTokenConverter (){
    /*utiliza hmacsha-256 simétrico*/
    JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
    jwtAccessTokenConverter.setSigningKey("algaworks");

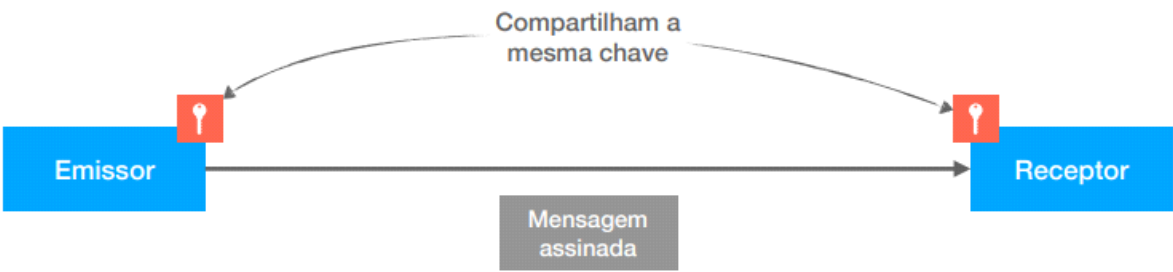
    return jwtAccessTokenConverter;
}
```

23.7. Entendendo a diferença entre assinatura com chave simétrica e assimétrica

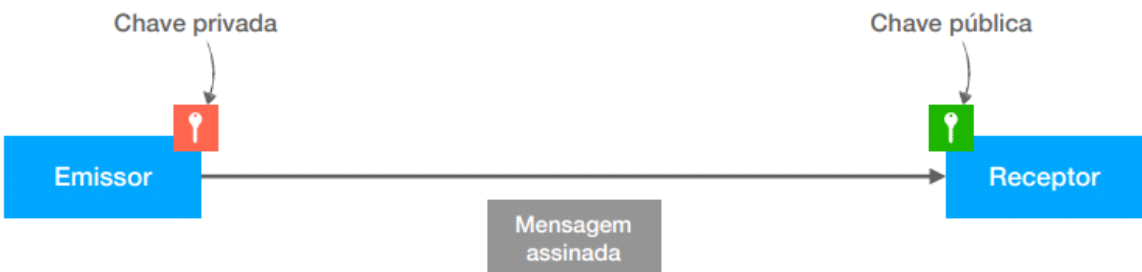
sábado, 13 de maio de 2023 17:44



Assinatura com chave simétrica



Assinatura com chave assimétrica



23.8. Gerando um par de chaves com keytool

sábado, 13 de maio de 2023 17:50

Como sabemos, para usar a assinatura com chaves assimétricas, temos que ter um conjunto de chaves públicas e privadas.

No java, temos a ferramenta KeyStore que trabalha com pares de chaves (pública e privada). Cria e gerencia arquivos .jks que funciona como repositório que dentro tem pares de chaves. Com a chave privada assinamos o token e com a pública validamos.

keytool -genkeypair -alias algafood -keyalg RSA -keypass 123456 -keystore algafood.jks -storepass 123456

o arquivo .jks pode conter vários pares de chaves e caso não tenha o arquivo algafood.jks, ele irá criar um novo, caso houver, será adicionada mais outros pares dentro do arquivo

keytool -genkeypair -alias algafood -keyalg RSA -keypass 123456 -keystore algafood.jks -storepass 123456 -validity 3650

Listando as entradas de um arquivo JKS

keytool -list -keystore algafood.jks

```

Administrador: Prompt de Comando
C:\Windows\system32>keytool -list -keystore algafood.jks
keytool error: java.lang.Exception: Keystore file does not exist: algafood.jks

C:\Windows\system32>keytool
Key and Certificate Management Tool

Commands:

-certreq          Generates a certificate request
-changealias      Changes an entry's alias
-delete          Deletes an entry
-exportcert       Exports certificate
-genkeypair       Generates a key pair
-genseckey        Generates a secret key
-gencert          Generates certificate from a certificate request
-importcert       Imports a certificate or a certificate chain
-importpass       Imports a password
-importkeystore   Imports one or all entries from another keystore
-keypasswd        Changes the key password of an entry
-list             Lists entries in a keystore
-printcert        Prints the content of a certificate
-printcertreq     Prints the content of a certificate request
-printcrl         Prints the content of a CRL file
-storepasswd      Changes the store password of a keystore
-showinfo         Displays security-related information

Use "keytool -?, -h, or --help" for this help message
Use "keytool -command_name --help" for usage of command name.
Use the -conf <url> option to specify a pre-configured options file.

C:\Windows\system32>keytool -genkeypair -alias algafood -keyalg RSA -keypass 123456 -keystore algafood.jks -storepass 123456
What is your first and last name?
[Unknown]:  Wenderson Santos
What is the name of your organizational unit?
[Unknown]:  AlgaFood
What is the name of your organization?
[Unknown]:  AlgaWorks
What is the name of your City or Locality?
[Unknown]:  Cametá
What is the name of your State or Province?
[Unknown]:  PA
What is the two-letter country code for this unit?
[Unknown]:  BR
Is CN=Wenderson Santos, OU=AlgaFood, O=AlgaWorks, L=Camet , ST=PA, C=BR correct?
[no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 90 days
for: CN=Wenderson Santos, OU=AlgaFood, O=AlgaWorks, L=Camet , ST=PA, C=BR

C:\Windows\system32>ls
'ls' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Windows\system32>keytool -list -keystore algafood.jks
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

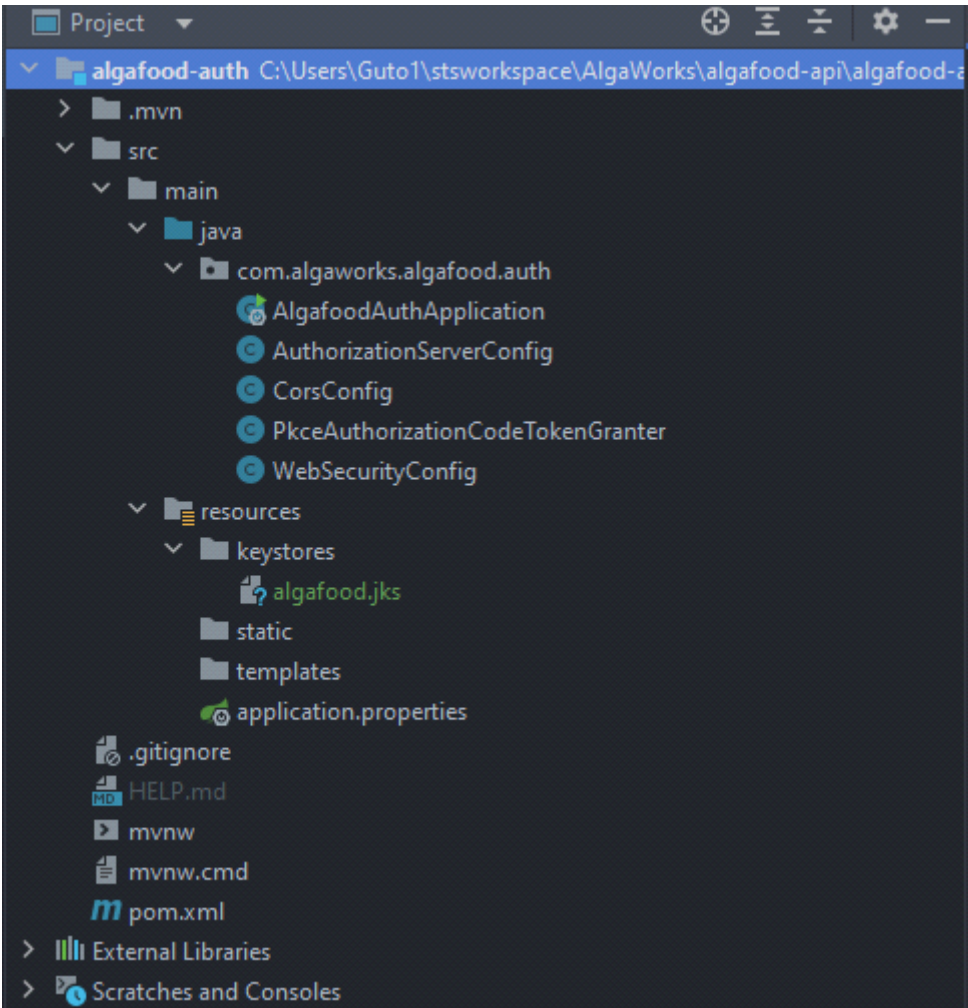
Your keystore contains 1 entry

algafood, 13 de mai. de 2023, PrivateKeyEntry,
Certificate fingerprint (SHA-256): 79:70:D3:97:DB:E0:30:1A:4A:D7:FF:BB:5B:2A:C2:AA:EA:97:FC:B2:B2:98:74:ED:D7:8D:D8:61:4D:DD:90:F5

C:\Windows\system32>
```


23.9. Assinando o JWT com RSA SHA-256 (chave assimétrica)

sábado, 13 de maio de 2023 18:02



Nota: Não utilizar o mesmo arquivo jks em produção

- dentro do arquivo tem um par de chaves.
 - Para chave pública: Validação
 - Para chave privada: Assinatura do token

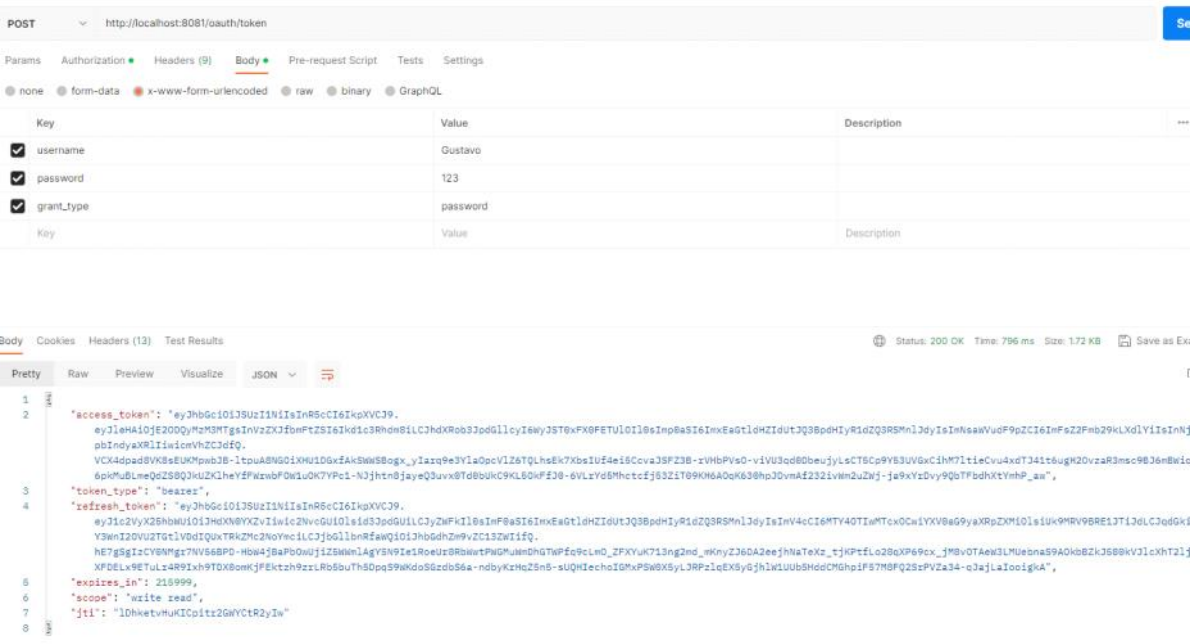
Nesta aula iremos emitir um token assinado com a chave privada

```
1 usage
@Bean/*Converte informações de user Logado para JWT * pode ser usado como Bean*/
public JwtAccessTokenConverter jwtAccessTokenConverter (){
    /*utiliza hmacsha-256 simétrico*/
    JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
    //jwtAccessTokenConverter.setSigningKey("ansbchd978234dkfjhsd98sd42nLkj2094kwejs0d8g");

    final ClassPathResource jksResource = new ClassPathResource("keystores/algafood.jks");
    String keyStorePass = "132456";
    String keyPairAlias = "algafood";

    final KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(jksResource, keyStorePass.toCharArray());
    final KeyPair keyPair = keyStoreKeyFactory.getKeyPair(keyPairAlias);
    jwtAccessTokenConverter.setKeyPair(keyPair);
    return jwtAccessTokenConverter;
}
```

até o momento o R.S não está configurado para receber um token jwt assimétrico, pois ele precisa da chave pública para a validação



23.10. Desafio: criando bean de propriedades de configuração do KeyStore

sábado, 13 de maio de 2023 19:48

23.11. Extraindo a chave pública no formato PEM

sábado, 13 de maio de 2023 20:44

- Extrair certificado no par de chaves, que contém a chave pública
- keytool -export -rfc -alias algafood -keystore algafood.jks -file algafood-cert.pem

pem é um formato de arquivo de certificado
gera um arquivo .pem que contém o certificado. A partir do arquivo que contém o certificado, extrair a chave pública

- openssl x509 -pubkey -noout -in algafood-cert.pem > algafood-pkey.pem

Caso o openssl esteja instalado na máquina

Outra opção é acessar pelo endpoint

```
/*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.
    security.checkTokenAccess("permitAll()");//permitindo qualquer cliente
    .tokenKeyAccess("permitAll()");//permite todos os acessos ao endpoint de acesso ao tokenkey publico do jwt
    .allowFormAuthenticationForClients();//permitindo autenticação de client em um form
    // P/ acessar o check token, precisa de autenticação do cliente
}
```

AlgaFood - Authorization Server / Validação Key Publica (JWT)

GET

▼

http://localhost:8081/oauth/token_key

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

	Key	Value
	Key	Value

BodyCookiesHeaders (14)Test Results

PrettyRawPreviewVisualizeJSON▼

1234

```
"alg": "SHA256withRSA",
"value": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQI
C09VJCX1900WIZmrr3U5rWUaknXba1aMXOX9cNLDXm09c1X+kxLtH1f/2x4Lh9/
QqyaeUVV+SdoQojzjaCUREhjifNwJpnCWVzNwNwYnUNaWK0wtgl0L1uegU52IPo
P1VoN63X9eU12ug1qZj30eCuafvHEpU49SSQppu
yAn-1TKCuPP7kgUzsvvLSb1a-
00ejWdwXHN4xEtzBVkKkqbu0mwgCCFd_ayn8yjf
a_th5hncbcbaFS11YDxM4Um-
UWh1ceALVpFOYUcFifRwf1lx0JfHghGoY0kMomN
xEuEC9uMeBM8rJs-0n03uCELVv4n-gSdt5zPNw
-----END PUBLIC KEY-----"

```

Private Key in PKCS #8, PKCS #1, or JWK string format. The key never leaves your browser.

RSASHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
-----BEGIN PUBLIC KEY-----MI
IBIjANBgkqhkiG9w0BAQEFAAOCAQ
8AMIIBCgKCAQEA/iopsjK8CgGE+
fB7DuASIZTLkabaUcpWVTRsUL1C2
)

Signature Verified

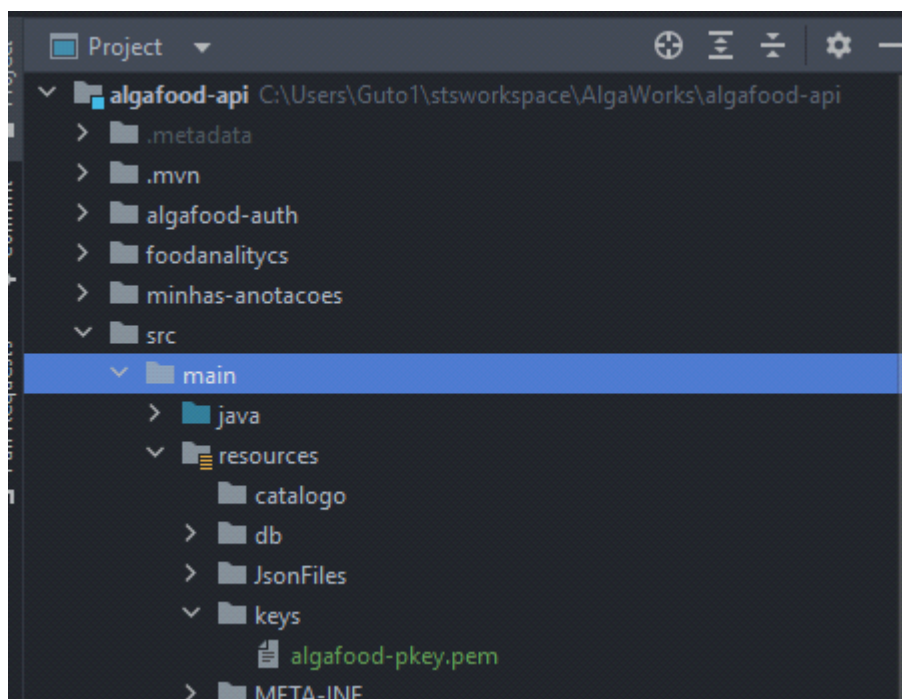
SHARE JWT

<https://jwt.io/>

Página 17 de 23. OAuth2 avançado com JWT e controle de acesso

23.12. Configurando a validação de JWT no Resource Server com a chave pública

sábado, 13 de maio de 2023 21:21



```
#23.12. Configurando a validação de JWT no Resource Server com a chave pública  
spring.security.oauth2.resourceserver.jwt.public-key-location=classpath:keys/algafood-pkey.pem
```


Agora basta o client fazer a requisição normalmente usando o Token JWT. Primeiro é necessário fazer a requisição em oauth/token e gerar o token de acesso usando as credenciais cadastradas no R.S (client id e secret)

AlgaFood - Authorization Server / Password Flow - 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"/> username	Gustavo
<input checked="" type="checkbox"/> password	123
<input checked="" type="checkbox"/> grant_type	password
Key	Value


POST	▼	http://localhost:8081/oauth/token				
Params	Authorization ●	Headers (9)	Body ●	Pre-request Script	Tests	Settings
Type	Basic Auth ▼		<i>i</i> Heads up! These parameters hold sensitive data. To keep this data variables ↗			
The authorization header will be automatically generated when you send the request. Learn more about authorization ↗			Username			
			algafood-web			
			Password			
			web123			

usamos o password flow por ser mais simples gerar o token de acesso.

GET 

http://localhost:8080/v1/cozinhas

Params 

Authorization 

Headers (8)


Body

Pre-request Script


Tests

Settings

Type

Bearer Token 

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

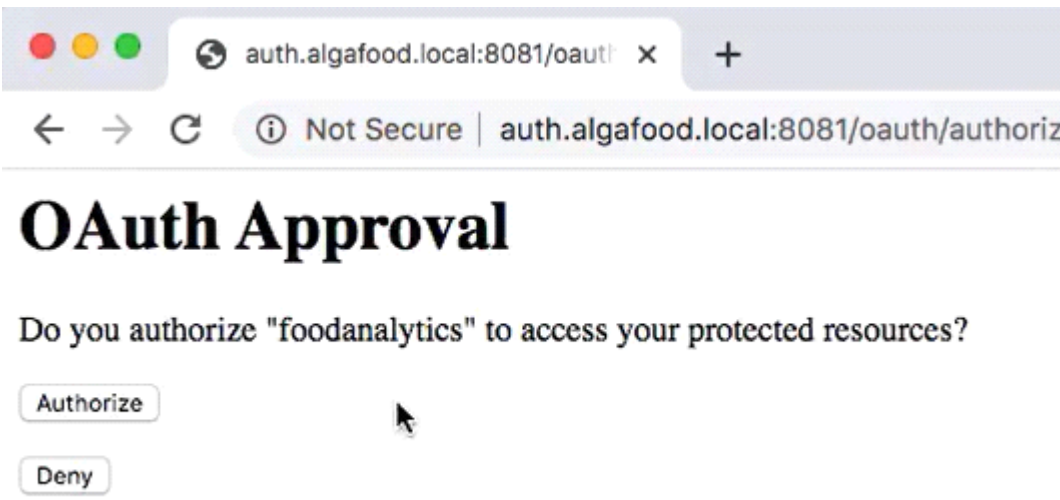
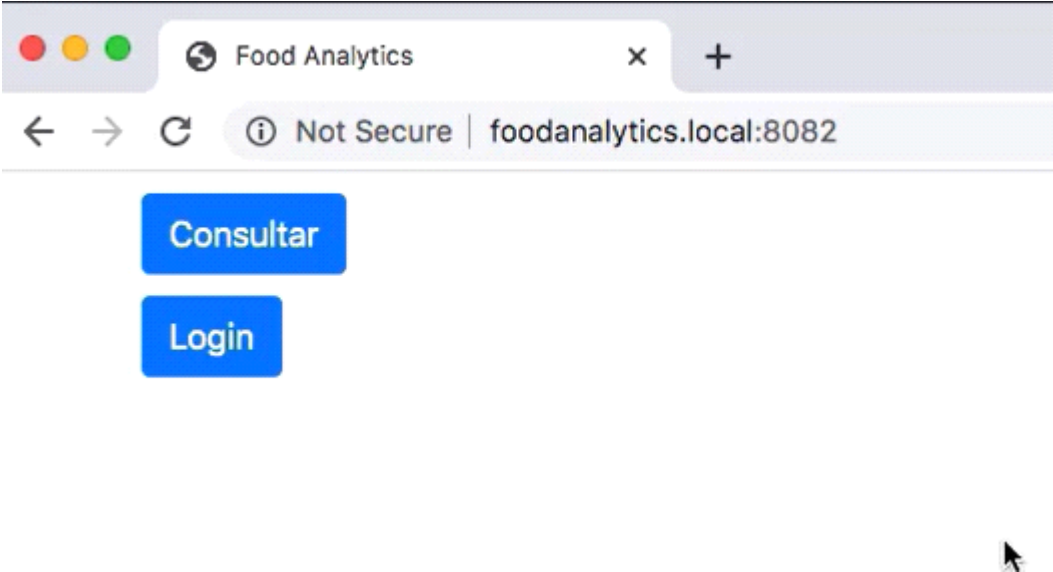
 Heads up! These parameters hold sensitive data. To keep this data secure while working in a collabora
[variables](#) ↗

Token

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

23.13. Revisando o fluxo de aprovação do Authorization Code com JWT

sábado, 13 de maio de 2023 22:24



Antes tinha escopos de forma individual, agora não tem mais graças ao jwt no A.S. O A.S configurou um handler que não tem mais os escopos

```
AuthorizationServerConfig.java
83
84 @Override
85 public void configure(AuthorizationServerEndpointsConfigurer endpoints)
86     endpoints
87         .authenticationManager(authenticationManager)
88         .userDetailsService(userDetailsService)
89         .reuseRefreshTokens(false)
90         .accessTokenConverter(jwtAccessTokenConverter())
91         .approvalStore(approvalStore(endpoints.getTokenStore()))
92         .tokenGranter(tokenGranter(endpoints));
93 }
94
95 private ApprovalStore approvalStore(TokenStore tokenStore) {
96     var approvalStore = new TokenApprovalStore();
97     approvalStore.setTokenStore(tokenStore);
98
99     return approvalStore;
100 }
101
```

é importante o approvalStore ser depois de accessTokenConverter

23.14. Autenticando usuário com dados do banco de dados

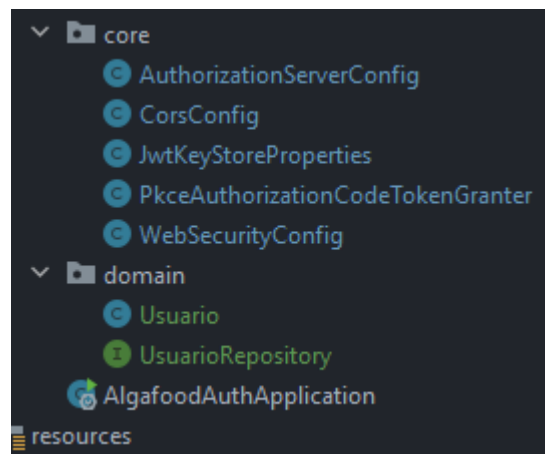
sábado, 13 de maio de 2023 22:40

Autenticar usuário final com dados do banco de dados. Até o momento estamos adicionando usuários em memória.

Nota: O servidor de Autorização ainda está separado da aplicação

- Adicionar o Spring Data JPA, Lombok e Driver Mysql no A.S
- Adicionar credenciais de acesso ao banco de dados no application.properties
- Adicionar entidade Usuário do R.S ao A.S

A ideia é utilizar os mesmo dados do banco de dados no A.S para utilizar os usuários e senhas como usuários finais no A.S



```

package com.algaworks.algafood.auth.domain;

import ...

@Entity
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Usuario {

    @Id
    @EqualsAndHashCode.Include
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String senha;

    @Column(nullable = false)
    private String email;
}

```

O repositório foi alterado para usar o JpaRepository, pois antes era uma implementação customizada.

- Criar classe que implementa UserDetailsService e anotar com @Service

```

package com.algaworks.algafood.auth.core;

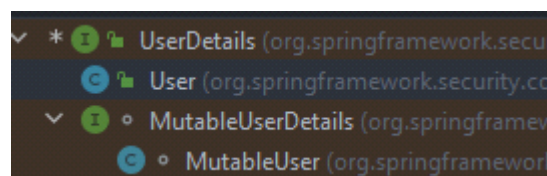
import ...

@Service
public class JpaUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return null;
    }
}

```

Dado um username, consultar no banco e retornar um objeto UserDetails. Precisamos de uma instância de UserDetails, que por sua vez é uma interface, ou seja, precisamos de uma classe que implemente UserDetails.



A classe User é uma implementação, mas não contém todas as informações necessárias para nossa aplicação, então teremos que estendê-la com uma implementação própria.


```

package com.algaworks.algafood.auth.core;

import ...

public class AuthUser extends User {
    @Serial
    private static final long serialVersionUID = -2094010971499823455L;
    1 usage
    private String fullName;

    public AuthUser (Usuario usuario){
        super(usuario.getEmail(), usuario.getSenha(), Collections.emptyList());

        fullName = usuario.getEmail();
    }
}

```

```

package com.algaworks.algafood.auth.core;

import ...

@Service
public class JpaUserDetailsService implements UserDetailsService {
    1 usage
    @Autowired
    private UsuarioRepository usuarioRepository;

    2 usages
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        /*O login é o email (username)*/
        Usuario usuario = usuarioRepository.findByEmail(username)
            .orElseThrow(() ->
                new UsernameNotFoundException("Usuário não encontrado com o email informado"));

        return new AuthUser(usuario);
    }
}

```

Adicionar Getter para a classe AuthUser para ter o get de fullName

- Retirar da classe WebSecurityConfig o método protected void configure(AuthenticationManagerBuilder auth), que configura usuários em memória
- Retirar da mesma classe o método protected UserDetails userDetailsService(), pois temos o nosso próprio UserDetailsService o JpaUserDetailsService

Adicionar no banco de dados as senhas dos usuários com Bcrypt

Send

Cookies

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

Bulk Edit

Status: 200 OK Time: 297 ms Size: 1.68 KB Save as Example

Q

1

23.15. Desafio: refatorando serviços de usuários para usar BCrypt

domingo, 14 de maio de 2023 10:59

23.16. Adicionando claims customizadas no payload do JWT

domingo, 14 de maio de 2023 13:16

Como visto na aula [23.4. Transparent Tokens: conhecendo o JSON Web Tokens \(JWT\)](#) o token é dividido em 3 partes, e uma delas contém claims (propriedades chave:valor) vamos customizar adicionando mais claims. Não é recomendado adicionar informações sensíveis.

é útil adicionar informações do usuário do token para evitar a consulta no banco de dados, ou o próprio cliente utilizar as informações no token para adicionar informações na página.

- Criar classe para customização

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

import ...

1 usage

public class JwtCustomClaimsTokenEnhancer implements TokenEnhancer {

    /**
     * customiza o token antes de ser emitido
     *
     * @param oAuth2AccessToken
     * representa o token e pode customizar o token antes de ser emitido
     * @param oAuth2Authentication
     * representa a autenticação do usuário, dentro contém o User
     * @return
     */
    @Override
    public OAuth2AccessToken enhance(OAuth2AccessToken oAuth2AccessToken, OAuth2Authentication oAuth2Authentication) {

        if(oAuth2Authentication.getPrincipal() instanceof AuthUser){
            final AuthUser authUser = (AuthUser) oAuth2Authentication.getPrincipal();

            final HashMap<String, Object> info = new HashMap<>();

            info.put("nome_completo", authUser.getFullName());
            info.put("usuario_id", authUser.getUserId());

            oAuth2AccessToken = (DefaultOAuth2AccessToken) oAuth2AccessToken;
            ((DefaultOAuth2AccessToken) oAuth2AccessToken).setAdditionalInformation(info);
        }

        return oAuth2AccessToken;
    }
}
```

O método `enhance` possui dois argumentos, `oAuth2AccessToken` que representa o próprio token que pode ser customizado no corpo do método, e o `oAuth2Authentication` que representa a autenticação (contém informações do usuário, token, e implementações)

foi necessário verificar se o retorno do método `oAuth2Authentication.getPrincipal()` é um `AuthUser` (extends `User`) pois existem outros tipos de fluxos de `oAuth2`, como o `client credentials` que não possuem usuário final e retornam outro tipo de fluxo.

o `OAuth2AccessToken` está tipado como interface (interface `OAuth2AccessToken`), logo, não temos o método `.setAdditionalInformation` que está em uma implementação dessa interface, a `DefaultOAuth2AccessToken`, temos que fazer o cast.

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    final TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
    tokenEnhancerChain.setTokenEnhancers(Arrays.asList(new JwtCustomClaimsTokenEnhancer(), jwtAccessTokenConverter()));

    endpoints.authenticationManager(authenticationManager). //somente o fluxo "password" precisa do authenticationManager, pois é assim que funciona o seu fluxo
    userDetailsService(userDetailsService). //para refresh_token*/
    reuseRefreshTokens(false).
    tokenEnhancer(tokenEnhancerChain).
    accessTokenConverter(jwtAccessTokenConverter()).
    approvalStore(approvalStore(endpoints.getTokenStore())).
    tokenGranter(tokenGranter(endpoints));
}
```

Agora temos que usar nossa implementação de TokenEnhancer na classe AuthorizationServerConfig, no método configure(AuthorizationServerEndpointsConfigurer endpoints)

Encoded

PASTE A TOKEN HERE

Decoded

EDIT THE PAYLOAD AND SECRET

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3VhcmIvX2lkIjoxLCJ1c2VyX25hbWUiOiJkb2FvLmdldkIjBhbGdhZm9vZC5jb20iLCJzY29wZSI6WyJ3cm10ZSI6InJlYWQiXSwiZXhwIjoxNjg0MzAzNjUyLCJqdGkiOiJxSWRHWXpncE12ZzBoWUM5ZU1fMVFqZ1lBRU0iLCJub211X2NvbXBsZXRvIjoiaW9hby5nZXJAYWxnYWZvb2QuY29tIiwiaWF0Ij01LqZmQjX1DspJ262aIZdbtoUyEJ_ufrI5tftdt4n88oKJaFYiqY0weM42pamsFoS88DS6FEeCwM5tfpVH3BbHoLoCV-qT1EzJap0SK5xFCqqbH26M1PYwhqgbAPJd4-ZFQ6BN4C9GJi3S1rsjSkyjPUVU4afz6uAdz00taP2IN2iZzI6H9mfjmx4oeGr2vLGRi13Mzt0PxQ0pP5aSeDsSuS2zndwPulELAFpZrI3j9B3JRN6UEAS6V92NB9ZeSkc2BPW5PPq9bK70J7yPgdcM7Fw8V1mDeKN-Y83ZvqssqNRFUft9H4SDprkmmMIwwudKXbxMN-qmGypT89VsF3aPg

Signature or encryption algorithm

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "usuario_id": 1,
  "user_name": "joao.ger@algafood.com",
  "scope": [
    "write",
    "read"
  ],
  "exp": 1684303652,
  "jti": "qIdGYzgpMvg0hYC9eI_1QjfyAEM",
  "nome_completo": "joao.ger@algafood.com",
  "client_id": "algafood-web"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1,
```

23.17. Obtendo usuário autenticado no Resource Server

domingo, 14 de maio de 2023 15:06

Durante a construção do controlador de pedidos, no método de adicionar pedidos, definimos que todo pedido ao ser adicionado, seria de id 1, pois ainda não tínhamos a implementação de um usuário autenticado no sistema para emitir pedidos de um usuário autenticado. Na aula vamos fazer essa implementação.

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

import ...

2 usages
@Component
public class AlgaSecurity {

    1 usage
    public Authentication getAuthentication(){
        return SecurityContextHolder.getContext().getAuthentication();
    }

    1 usage
    public Long getUsuarioId() {

        final Jwt jwt = (Jwt) getAuthentication().getPrincipal();

        return jwt.getClaim("usuario_id");
    }
}
```

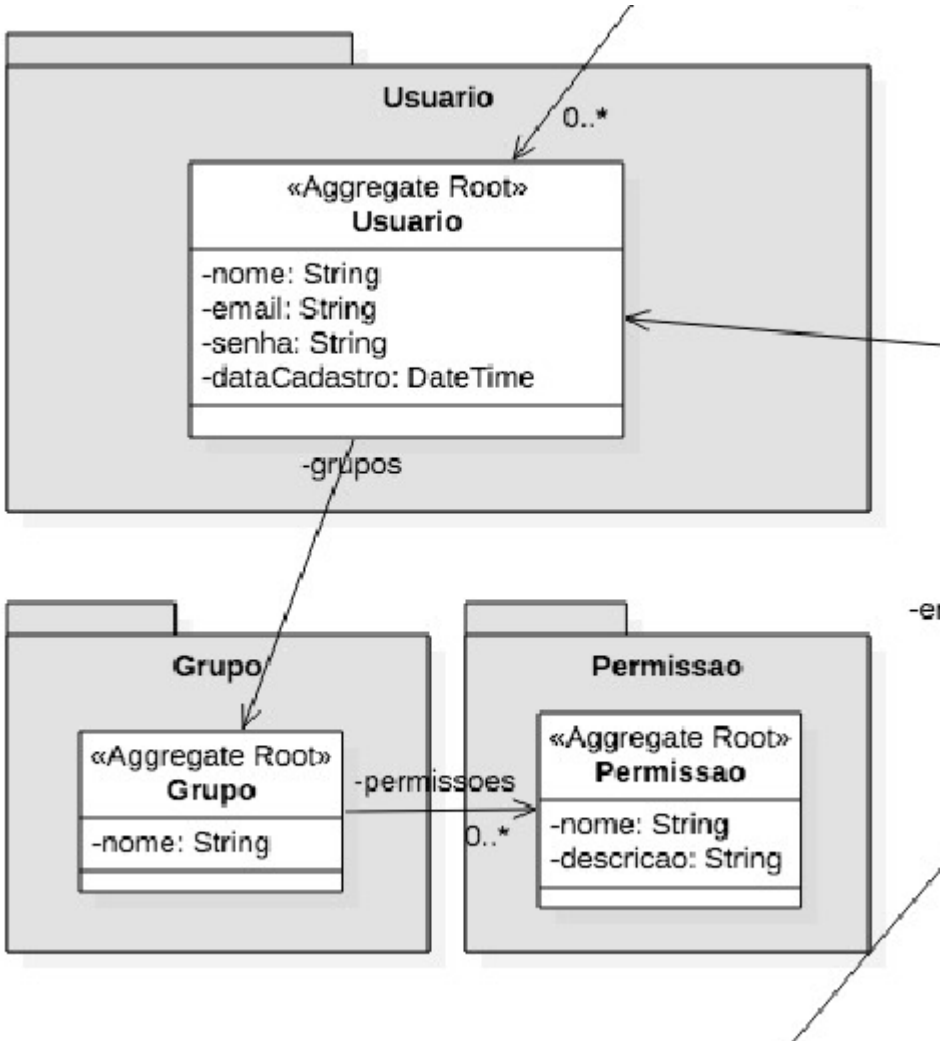
o claim é definido na aula 23.16

e a classe é usada no controlador de pedido, para adicionar o pedido ao usuário autenticado

```
1 usage
private void validarPedido(Pedido pedido) {
    pedido.setCliente(new Usuario());
    pedido.getCliente().setId(algaSecurity.getUsuarioId());
}
```

23.18. Definindo e criando as permissões de acesso

domingo, 14 de maio de 2023 15:40



Há uma tabela de permissões que está associada a um grupo, e o usuário está associado a um grupo, com isso, definimos as restrições de acesso ao sistema por permissões de acesso.

23.19. Carregando as permissões concedidas na autenticação

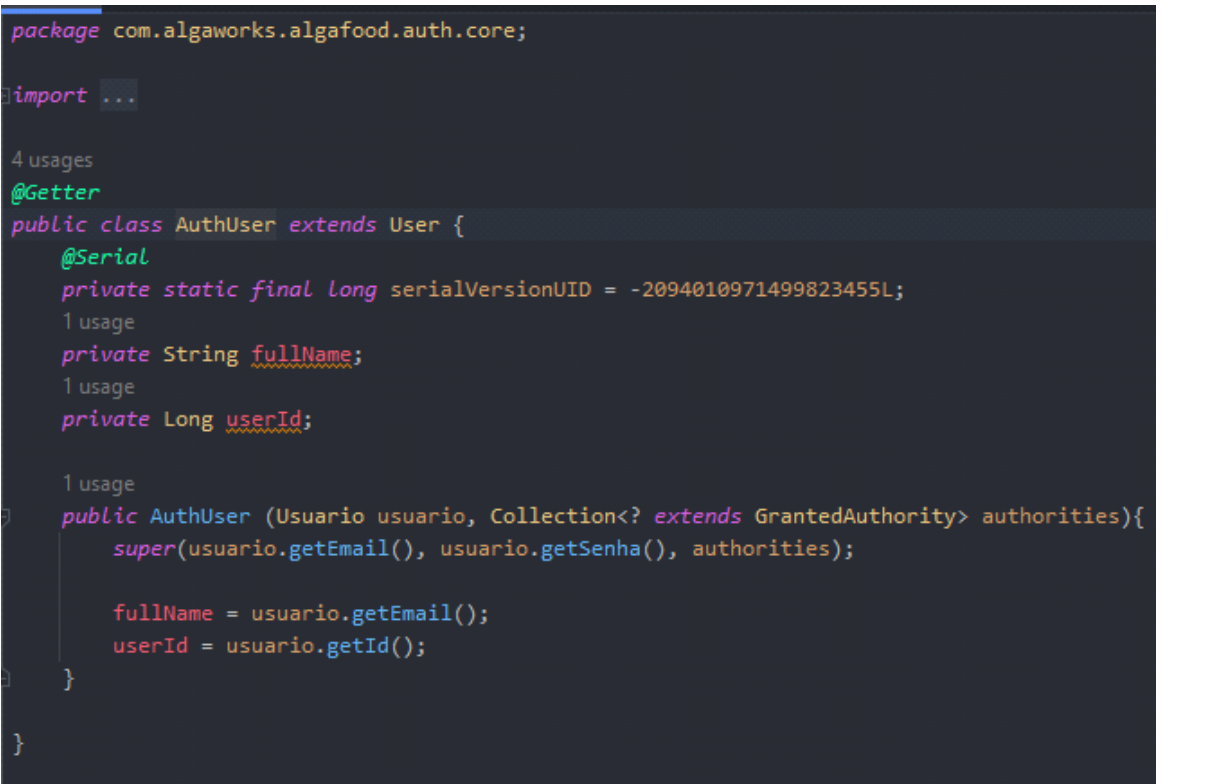
domingo, 14 de maio de 2023 15:54

Carregar as permissões de um usuário na emissão de um token. Pois nesse momento, não estamos tratando as permissões do usuário na emissão do token. Já existe as permissões e grupos no banco de dados, vamos apenas gerenciar isso no token.

No A.S adicionar classes que fazem sentido para as permissões do usuário, estamos reutilizando as tabelas e classes do R.S

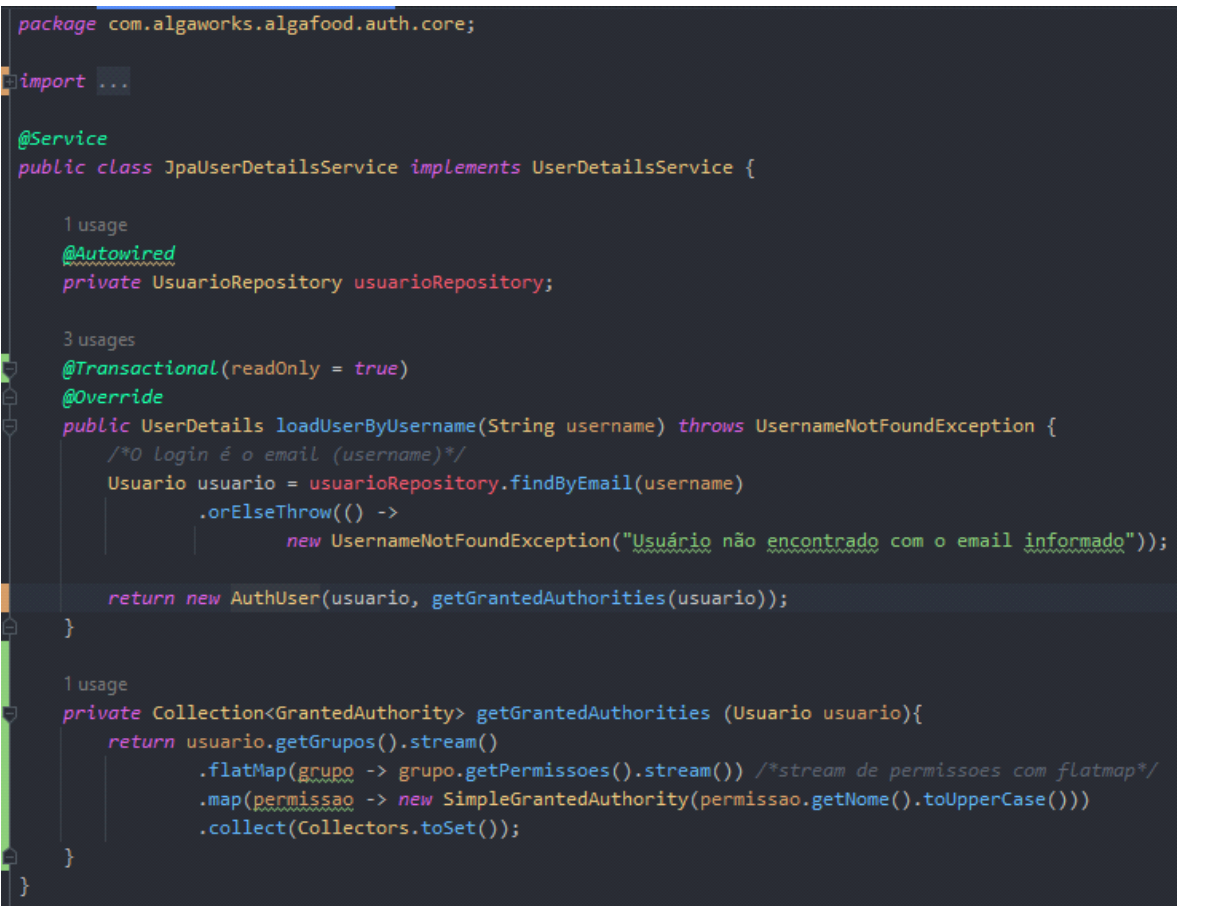


na classe AuthUser



Adicionamos um argumento que recebe uma coleção de GrantedAuthority e repassando para a superclasse

Na JpaUserDetailsService



23.20. Carregando as Granted Authorities e restringindo acesso a endpoints na API

domingo, 14 de maio de 2023 17:40

Atualmente, no R.S está configurado para que qualquer requisição autenticada pode acessar qualquer coisa na api

```
1 package com.algaworks.algafood.core.security;
2
3 import org.springframework.context.annotation.Configuration;
4
5 @Configuration
6 @EnableWebSecurity
7 public class ResourceServerConfig extends WebSecurityConfigurerAdapter {
8
9     @Override
10     protected void configure(HttpSecurity http) throws Exception {
11         http
12             .authorizeRequests()
13                 .anyRequest().authenticated()
14             .and()
15             .cors().and()
16             .oauth2ResourceServer().jwt();
17     }
18 }
19 }
```

Então, dada as restrições de permissões configuradas na autenticação e adicionada à claim "authorities" como na aula [23.16. Adicionando claims customizadas no payload do JWT](#)

podemos configurar o acesso aos endpoints para determinado tipo de permissão de um usuário

Fazemos essa configuração no R.S. na classe ResourceServerConfig

```
@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()
            .antMatchers(HttpMethod.POST, ...antPatterns: "/v1/cozinhas/**").hasAuthority("EDITAR_COZINHAS")
            .antMatchers(HttpMethod.PUT, ...antPatterns: "/v1/cozinhas/**").hasAuthority("EDITAR_COZINHAS")
            .antMatchers(HttpMethod.GET, ...antPatterns: "/v1/cozinhas/**").authenticated()
            .anyRequest().denyAll()
            .and()
            .cors()
            .and()
            .oauth2ResourceServer()
                .opaqueToken();
            .jwt()
            .jwtAuthenticationConverter(jwtAuthenticationConverter());
}
```

as request POST, PUT só podem ser acessadas por um usuário que conter a permissão "EDITAR_COZINHAS" e para GET, basta o usuário estar autenticado.

As demais requisições não são permitidas, com a configuração `anyRequests().denyAll()`

```
48 @ private JwtAuthenticationConverter jwtAuthenticationConverter(){
49     final JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();
50
51     jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(jwt -> {
52         List<String> authorities = jwt.getClaimAsStringList("authorities");
53
54         if(authorities == null)
55             authorities = Collections.emptyList();
56
57         return authorities.stream()
58             .map(SimpleGrantedAuthority::new)
59             .collect(Collectors.toList());
60     });
61
62     return jwtAuthenticationConverter;
63 }
64 }
```


23.21. Method Security: Restringindo acesso com @PreAuthorize e SpEL

domingo, 14 de maio de 2023 18:25

- Retirar as configurações de restrições da aula anterior do R.S

```
@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
        .cors().and()
        .csrf().disable()
        .oauth2ResourceServer()
            .opaqueToken();
        .jwt()
        .jwtAuthenticationConverter(jwtAuthenticationConverter());
}
```

```
oauth2ResourceServer().opaqueToken()
    * (opaque token) para autenticação do usuário.*/
    http
        .cors().and()
        .csrf().and()
        .oauth2ResourceServer()
            .opaqueToken();
        .jwt()
```

Desabilitar csrf para POST

neste momento pode acessar qualquer requisição sem token.

@EnableGlobalMethodSecurity

```
3      import ...
14
15      @Configuration
16      @EnableWebSecurity
17      @EnableGlobalMethodSecurity(prePostEnabled = true)
18      public class ResourceServerConfig extends WebSecurityConfigurerAdapter {
19
```

Isso no R.S

agora temos que configurar os métodos no controlador

```
Config.java x CozinhaController.java x
1 usage
@Autowired
private PagedResourcesAssembler<Cozinha> pagedResourcesAssembler;

@PreAuthorize("hasAuthority('EDITAR_COZINHAS')")
@Override
@PostMapping
public ResponseEntity<CozinhaDTO> adicionar(@RequestBody @Valid CozinhaInputDTO cozinhaInputDTO) {
    ...
}

@PreAuthorize("isAuthenticated()")
@Override
@GetMapping
public PagedModel<CozinhaDTO> listar(@PageableDefault(size = 5) Pageable pageable) {
    ...
}

@PreAuthorize("isAuthenticated()")
@Override
@GetMapping("/{cozinhaId}")
public CozinhaDTO buscar(@PathVariable Long cozinhaId) {
    ...
}

@PreAuthorize("hasAuthority('EDITAR_COZINHAS')")
@Override
@PutMapping("/{cozinhaId}")
public CozinhaDTO atualizar(@RequestBody @Valid CozinhaInputDTO cozinhaInputDTO, @PathVariable Long cozinhaId) {
    ...
}

@PreAuthorize("hasAuthority('EDITAR_COZINHAS')")
@Override
@DeleteMapping("/{cozinhaId}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deletar(@PathVariable Long cozinhaId) {
    cozinhaService.deletar(cozinhaId);
}
}
```

23.22. Desafio: tratando AccessDeniedException no ExceptionHandler

domingo, 14 de maio de 2023 19:52

1 - Alterando ProblemType

Vamos adicionar mais um tipo no nosso enum ProblemType, chamaremos o mesmo de ACESSO_NEGADO

```
ACESSO_NEGADO("/acesso-negado", "Acesso negado"),
```

2 - Alterando ApiExceptionHandler

Agora vamos adicionar um novo método no nosso ExceptionHandler para tratarmos exceptions do tipo AccessDeniedException.

```
@ExceptionHandler(AccessDeniedException.class)
public ResponseEntity<?> handleEntidadeNaoEncontrada(AccessDeniedException
ex, WebRequest request) {

    HttpStatus status = HttpStatus.FORBIDDEN;
    ProblemType problemType = ProblemType.ACESSO_NEGADO;
    String detail = ex.getMessage();

    Problem problem = createProblemBuilder(status, problemType, detail)
        .userMessage(detail)
        .userMessage("Você não possui permissão para executar essa operação.")
        .build();

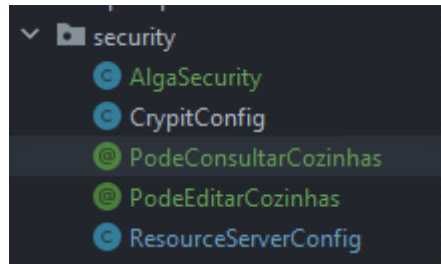
    return handleExceptionInternal(ex, problem, new HttpHeaders(), status,
request);
}
```

23.23. Simplificando o controle de acesso em métodos com meta-anotações

domingo, 14 de maio de 2023 19:57

Como na aula 23.21 criamos anotações para cada método a fim de restringir acessos a determinados usuários com determinadas permissões.

Podemos utilizar meta-anotações, que encapsulam as anotações para ter um único ponto de alteração caso necessário.



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

import ...

3 usages
@PreAuthorize("isAuthenticated()")
@Retention(RetentionPolicy.RUNTIME) //lida em tempo de exce
@Target(ElementType.METHOD) //apenas em métodos
public @interface PodeConsultarCozinhas {
}
```

```

@PodeEditarCozinhas
@Override
@PostMapping
public ResponseEntity<CozinhaDTO> adicionar(

@PodeConsultarCozinhas
@Override
@GetMapping
public PagedModel<CozinhaDTO> listar(@Page

@PodeConsultarCozinhas
@Override
@GetMapping("/{cozinhaId}")
public CozinhaDTO buscar(@PathVariable Long co

@PodeEditarCozinhas
@Override
@PutMapping("/{cozinhaId}")
public CozinhaDTO atualizar(@RequestBody Co

@PodeEditarCozinhas
@Override
@DeleteMapping("/{cozinhaId}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deletar(@PathVariable Long co

```

Ou podemos acoplar mais ainda para melhorar a organização

```

package com.algaworks.algafood.core.security;

import org.springframework.security.access.prepost.PreAuthorize;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

6 usages
public @interface CheckSecurity {

    5 usages
    @interface Cozinhas {

        2 usages
        @PreAuthorize("isAuthenticated()")
        @Retention(RetentionPolicy.RUNTIME) //lida em tempo de exce
        @Target(ElementType.METHOD) //apenas em métodos
        @interface PodeConsultar {}

        3 usages
        @PreAuthorize("hasAuthority('EDITAR_COZINHAS')")
        @Retention(RetentionPolicy.RUNTIME) //lida em tempo de exce
        @Target(ElementType.METHOD) //apenas em métodos
        @interface PodeEditar {}

    }
}

```

```

1 usage
@Autowired
private PagedResourcesAssembler<Cozinha> pagedResource

@CheckSecurity.Cozinhas.PodeEditar
@Override
@PostMapping
public ResponseEntity<CozinhaDTO> adicionar(@RequestBody CozinhaDTO cozinha) {
    return ResponseEntity.ok(pagedResource.add(cozinha));
}

@CheckSecurity.Cozinhas.PodeConsultar
@Override
@GetMapping
public PagedModel<CozinhaDTO> listar(@PageableDefault Pageable pageable) {
    return pagedResource.list(pageable);
}

@CheckSecurity.Cozinhas.PodeConsultar
@Override
@GetMapping("/{cozinhaId}")
public CozinhaDTO buscar(@PathVariable Long cozinhaId) {
    return pagedResource.get(cozinhaId);
}

@CheckSecurity.Cozinhas.PodeEditar
@Override
@PutMapping("/{cozinhaId}")
public CozinhaDTO atualizar(@RequestBody @Valid CozinhaDTO cozinha) {
    return pagedResource.update(cozinhaId, cozinha);
}

@CheckSecurity.Cozinhas.PodeEditar
@Override
@DeleteMapping("/{cozinhaId}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deletar(@PathVariable Long cozinhaId) {
    pagedResource.delete(cozinhaId);
}

```

23.24. Entendendo os escopos do OAuth2

domingo, 14 de maio de 2023

20:13

```
AuthorizationServerConfig.java 23
46         .withClient("algafood-web")
47         .secret(passwordEncoder.encode("web123"))
48         .authorizedGrantTypes("password", "refresh_token")
49         .scopes("write", "read")
50         .accessTokenValiditySeconds(6 * 60 * 60) // 6 horas
51         .refreshTokenValiditySeconds(60 * 24 * 60 * 60) // 60 dias
52
53     .and()
54     .withClient("foodanalytics")
```

O escopo inclusive pode ser qualquer nome.

Temos que configurar tanto no A.S quanto no R.S. A ideia é limitar o acesso de aplicações clientes. Mesmo que o usuário possua permissões para alterar e ler os recursos protegidos do R.S, ainda há uma camada de limitação pelo cliente que está acessando os recursos protegidos em nome do Resource Owner, que pode ser definido pelos escopos. Cada escopo é configurado tanto no R.S quanto no A.S.

23.25. Carregando Granted Authorities dos escopos do OAuth2 no Resource Server

domingo, 14 de maio de 2023 20:29

Quando o token não está configurando com Grated Authorities customizadas do claim customizado, as Authorities padrão são escopos. Logo, quando há configuração de Granted Authorities, o escopos não estão disponíveis no contextoHolder utilizando SecurityContextHolder.getContext().getAuthentication().getAuthorities.

SecurityContextHolder.getContext().getAuthentication().getAuthorities

retorna Granted Authorities:
quando há configuração de JwtAuthenticationConverter

```
        .jwt()
        .jwtAuthenticationConverter(jwtAuthenticationConverter());
    }

1 usage
private JwtAuthenticationConverter jwtAuthenticationConverter() {
    final JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();

    jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(jwt -> {
        List<String> authorities = jwt.getClaimAsStringList("authorities");

        if (authorities == null)
            authorities = Collections.emptyList();

        return authorities.stream()
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
    });

    return jwtAuthenticationConverter;
}
```

Retorna escopos:
Quando não há configuração de JwtAuthenticationConverter

Estamos na verdade substituindo a implementação padrão do JwtJwtAuthenticationConverter setando nosso jwt manualmente, enquanto antes era tudo automático

```
2 usages
public void setJwtGrantedAuthoritiesConverter(
    Converter<Jwt, Collection<GrantedAuthority>> jwtGrantedAuthoritiesConverter) {
    Assert.notNull(jwtGrantedAuthoritiesConverter, message: "jwtGrantedAuthoritiesConverter cannot be null");
    this.jwtGrantedAuthoritiesConverter = jwtGrantedAuthoritiesConverter;
}
```

refatoramos o código onde capturamos as authorities e adicionamos authorities dos escopos também

```
1 usage
private JwtAuthenticationConverter jwtAuthenticationConverter() {
    final JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter();

    jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(jwt -> {
        List<String> authorities = jwt.getClaimAsStringList("authorities");

        if (authorities == null)
            authorities = Collections.emptyList();

        final JwtGrantedAuthoritiesConverter scopesJwtGrantedAuthoritiesConverter = new JwtGrantedAuthoritiesConverter();
        final Collection<GrantedAuthority> grantedAuthorities = scopesJwtGrantedAuthoritiesConverter.convert(jwt);

        grantedAuthorities.addAll(authorities.stream()
            .map(SimpleGrantedAuthority::new).toList());
        return grantedAuthorities;
    });

    return jwtAuthenticationConverter;
}
```

Antes de listarmos as authorities dentro da claim "authorities" convertemos as authorities existentes, ou seja, escopos, para uma outra coleção de Authorities e depois adicionados o restante das authorities da claim.

```
2023-05-15 00:08:51.312 INFO 9036 --- [nio-8080-exec-5] c.a.a.v1.controller.CozinhaController : Quantidade de cozinhas 5
Hibernate: select cozinha0_.id as id1_1_, cozinha0_.nome as nome2_1_ from cozinha cozinha0_ limit ?
[SCOPE_write, SCOPE_read, EDITAR_COZINHAS, CONSULTAR_RESTAURANTES, EDITAR_FORMAS_PAGAMENTO, EDITAR_RESTAURANTES, CONSULTAR_USUARIOS, EDITAR_USUARIOS, CONSULTAR
```

final Collection<? extends GrantedAuthority> authorities =
SecurityContextHolder.getContext().getAuthentication().getAuthorities();

System.out.println(authorities);

23.26. Restringindo acesso a endpoints por escopos do OAuth2

domingo, 14 de maio de 2023 21:26

Até agora estamos restringindo o acesso a endpoint e métodos através das authorities da claim. Agora vamos implementar a restrições das authorities de escopo na geração de token, principalemten no fluxo Authorization Code [22.18. Configurando o Authorization Code Grant Type](#)

Vamos checar no escopo no R.S.

```
6 usages
public @interface CheckSecurity {

    5 usages
    @interface Cozinhas {

        2 usages
        @PreAuthorize("hasAuthority('SCOPE_READ') and isAuthenticated()")
        @Retention(RetentionPolicy.RUNTIME)//lida em tempo de exce
        @Target(ElementType.METHOD) //apenas em métodos
        @interface PodeConsultar {}

        3 usages
        @PreAuthorize("hasAuthority('SCOPE_WRITE') and hasAuthority('EDITAR_COZINHAS')")
        @Retention(RetentionPolicy.RUNTIME)//lida em tempo de exce
        @Target(ElementType.METHOD) //apenas em métodos
        @interface PodeEditar {}

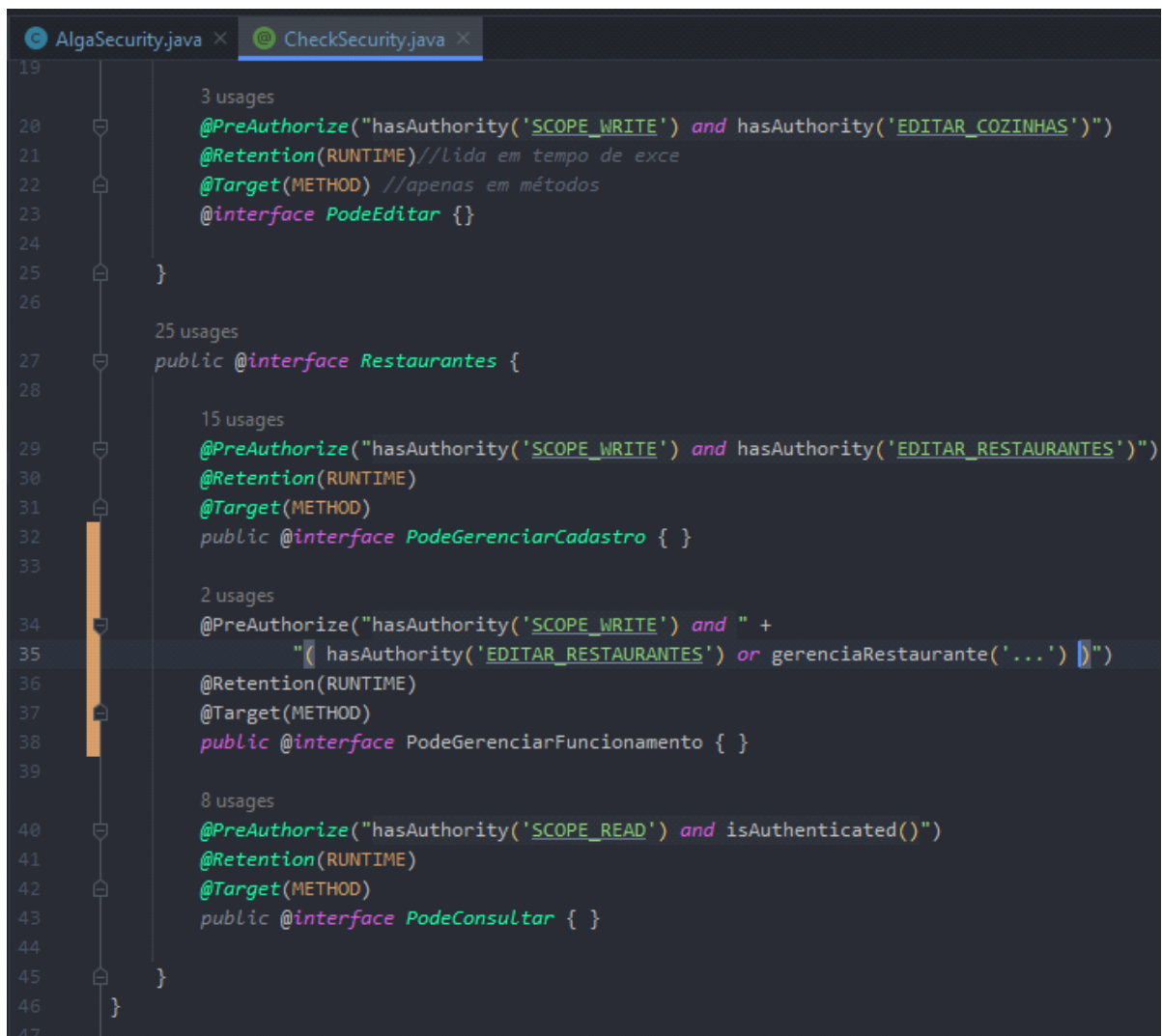
    }
}
```

23.28. Restringindo acessos de forma contextual (sensível à informação)

segunda-feira, 15 de maio de 2023 09:38

Exemplo: Usuário de id 5 gerencia os Restaurantes de id 1 e 3, mas o Usuário não tem nenhuma permissão para gerenciar restaurantes, e não podemos atribuir a permissão de "EDITAR_RESTAURANTES" pois ele vai poder gerenciar todos, e não apenas os que ele próprio gerencia.

O responsável do restaurante só pode gerenciar seus próprios restaurantes, e não restaurantes de outros responsáveis.



```
19
20 3 usages
21 @PreAuthorize("hasAuthority('SCOPE_WRITE') and hasAuthority('EDITAR_COZINHAS')")
22 @Retention(RUNTIME) //lida em tempo de exce
23 @Target(METHOD) //apenas em métodos
24 @interface PodeEditar {}
25
26
27 25 usages
28 public @interface Restaurantes {
29
30 15 usages
31 @PreAuthorize("hasAuthority('SCOPE_WRITE') and hasAuthority('EDITAR_RESTAURANTES')")
32 @Retention(RUNTIME)
33 @Target(METHOD)
34 public @interface PodeGerenciarCadastro { }
35
36 2 usages
37 @PreAuthorize("hasAuthority('SCOPE_WRITE') and " +
38 "({ hasAuthority('EDITAR_RESTAURANTES') or gerenciaRestaurante('...') })")
39 @Retention(RUNTIME)
40 @Target(METHOD)
41 public @interface PodeGerenciarFuncionamento { }
42
43 8 usages
44 @PreAuthorize("hasAuthority('SCOPE_READ') and isAuthenticated()")
45 @Retention(RUNTIME)
46 @Target(METHOD)
47 public @interface PodeConsultar { }
```

faremos outra anotação para verificar se o usuário autenticado tem permissão para editar todos os restaurantes (usuário adm) ou editar(gerenciar) restaurantes que ele mesmo gerencia (dono de restaurantes cadastrados no sistema)

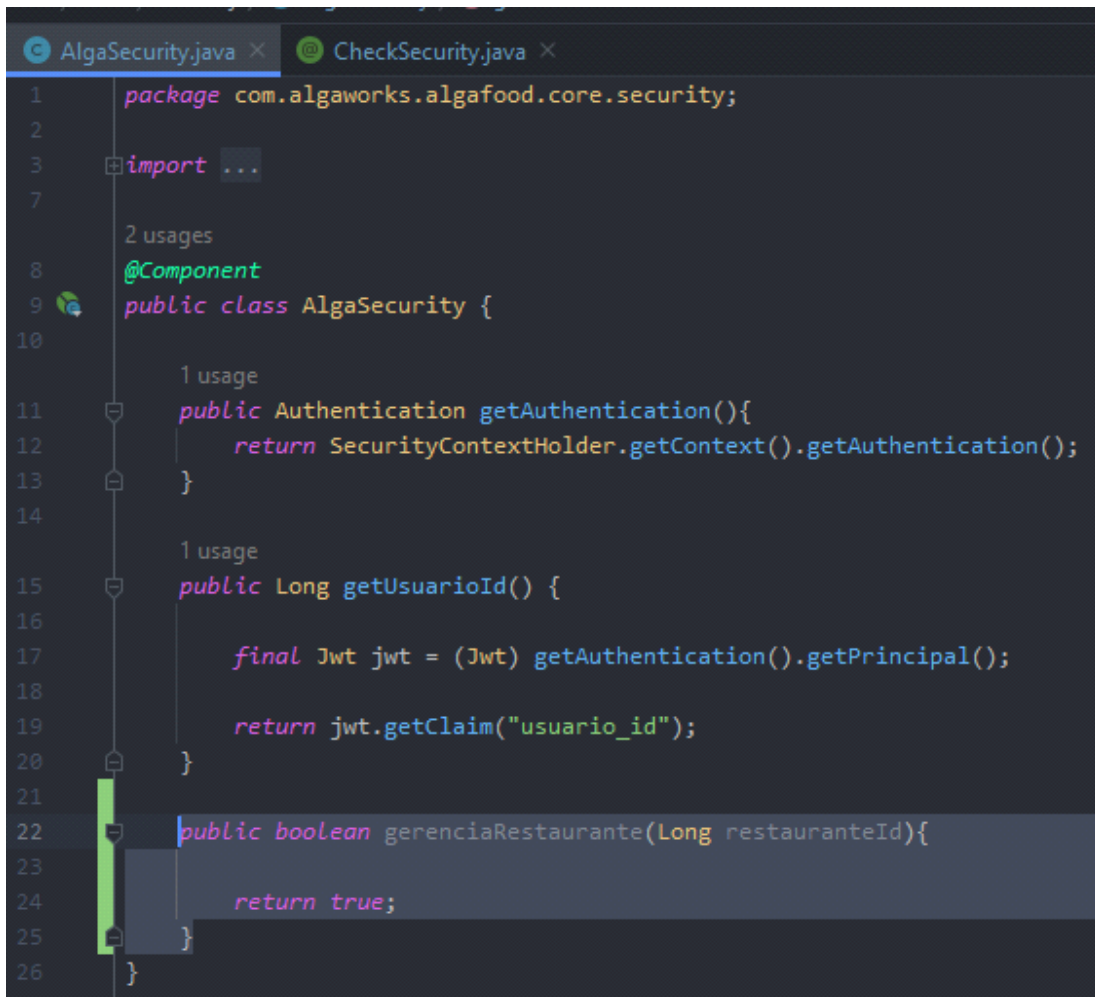
O escopo é de escrita? sim.

O usuário autenticado tem permissão para editar todos os restaurantes? não

O usuário autenticado tem permissão para editar apenas os restaurantes que ele toma de conta? sim.

e vice-versa.

Vamos implementar o "gerenciaRestaurante"



```
1 package com.algaworks.algafood.core.security;
2
3 import ...
4
5 2 usages
6
7 @Component
8 public class AlgaSecurity {
9
10 1 usage
11 public Authentication getAuthentication(){
12     return SecurityContextHolder.getContext().getAuthentication();
13 }
14
15 1 usage
16 public Long getUsuarioId() {
17     final Jwt jwt = (Jwt) getAuthentication().getPrincipal();
18     return jwt.getClaim("usuario_id");
19 }
20
21
22 public boolean gerenciaRestaurante(Long restauranteId){
23
24     return true;
25 }
26 }
```

Baseado no usuário autenticado, vamos aplicar a lógica para verificar se ele tem permissão de gerenciar seus próprios restaurantes. A ideia é restringir acesso a restaurantes que ele não gerencia. Somente o usuário autenticado em específico vai gerenciar seus restaurantes.

Vamos chamar o método `gerenciaRestaurante` passando o id do restaurante para verificar se o usuário autenticado gerencia o restaurante do id

Podemos, a partir de um `@PreAuthorize`, chamar um método que retorna boolean para fazer validações

```

25 usages
public @interface Restaurantes {

    15 usages
    @PreAuthorize("hasAuthority('SCOPE_WRITE') and hasAuthority('EDITAR_RESTAURAN
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodeGerenciarCadastro { }

    2 usages
    @PreAuthorize("hasAuthority('SCOPE_WRITE') and "
        + "( hasAuthority('EDITAR_RESTAURANTES') or"
        + "@algaSecurity. gerenciaRestaurante(Long restauranteId) boolean
        authentication Authentication
        usuarioId Long
        Press Enter to insert, Guia to replace
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface Pod

    8 usages
    @PreAuthorize("hasAuthority('SCOPE_READ') and isAuthenticated()")
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodeConsultar { }

}

```

Nota: a classe tem que ser um componente gerenciado pelo contexto do Spring e na realidade estamos chamando um bean, o nome do bean de uma classe gerenciada é o mesmo nome da classe, porém, iniciada com minúscula, por isso é "@algaSecurity".

```

2 usages
@PreAuthorize("hasAuthority('SCOPE_WRITE') and "
    + "( hasAuthority('EDITAR_RESTAURANTES') or"
    + "@algaSecurity.gerenciaRestaurante(#restauranteId) )"
@Retention(RUNTIME)
@Target(METHOD)
public @interface PodeGerenciarFuncionamento { }

```

```

1 usage
@CheckSecurity.Restaurantes.PodeGerenciarFuncionamento
@Override
@PutMapping("/{restauranteId}/abertura")
public ResponseEntity<Void> abrir(@PathVariable Long restauranteId) {...}

```

estamos chamando o bean (instância gerenciada da classe AlgaSecurity), chamamos o método e passamos como argumento o id do restaurante da classe controladora, a variável restauranteId é um pathVariable e pode ser passado como argumento usado #

Dentro de algaSecurity

```

1 usage
public boolean gerenciaRestaurante(Long restauranteId){

    return restauranteRepository.existsResponsavel(restauranteId, getUsuarioId());
}

```

a implementação da query ficou no orm.xml [5.10. Externalizando consultas JPQL para um arquivo XML](#)

```

<named-query name="Restaurante.existsResponsavel">
    <query>
        select case when count(1) > 0 then true else false end
        from Restaurante rest
        join rest.responsaveis resp
        where rest.id = :restauranteId
        and resp.id = :usuarioId
    </query>
</named-query>

```

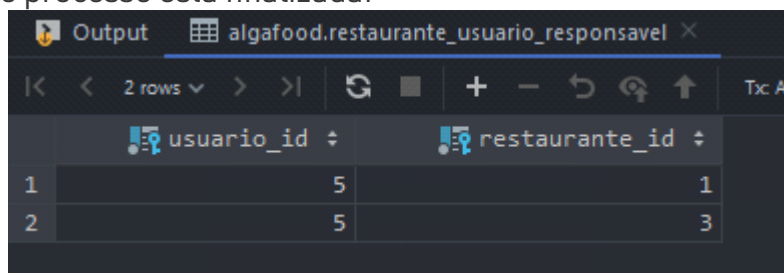
equivalente a

```

@Query("select case when count(1) > 0 then true else false end"
+ "    from Restaurante rest"
+ "    join rest.responsaveis resp"
+ "    where rest.id = :restauranteId"
+ "    and resp.id = :usuarioId")
boolean existsResponsavel(Long restauranteId, Long usuarioId);

```

A lógica do processo está finalizada.



	usuario_id	restaurante_id
1	5	1
2	5	3

no banco de dados, o usuário 5 gerencia o restaurante 1 e 3, logo, o usuário 5 só deverá gerenciar os restaurantes 1 e 3 e apenas esses.

No método controlador de restaurantes, temos as seguintes PreAuthorize

```

1 usage
@CheckSecurity.Restaurantes.PodeGerenciarCadastro
@Override
@DeleteMapping("/{restauranteId}/inativar")
public ResponseEntity<Void> inativar(@PathVariable Long restauranteId) {...}

1 usage
@CheckSecurity.Restaurantes.PodeGerenciarFuncionamento
@Override
@PutMapping("/{restauranteId}/abertura")
public ResponseEntity<Void> abrir(@PathVariable Long restauranteId) {...}

1 usage
@CheckSecurity.Restaurantes.PodeGerenciarFuncionamento
@Override
@PutMapping("/{restauranteId}/fechamento")
public ResponseEntity<Void> fechar(@PathVariable Long restauranteId) {...}

@CheckSecurity.Restaurantes.PodeGerenciarCadastro
@Override
@PutMapping("/ativacoes")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void ativarRestaurantes(@RequestBody Set<Long> restauranteIds) {...}

@CheckSecurity.Restaurantes.PodeGerenciarCadastro
@Override
@DeleteMapping("/inativacoes")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void inativarRestaurantes(@RequestBody Set<Long> restauranteIds) {...}

```

os métodos "abrir" e "fechar" são anotados com "PodeGerenciarFuncionamento"

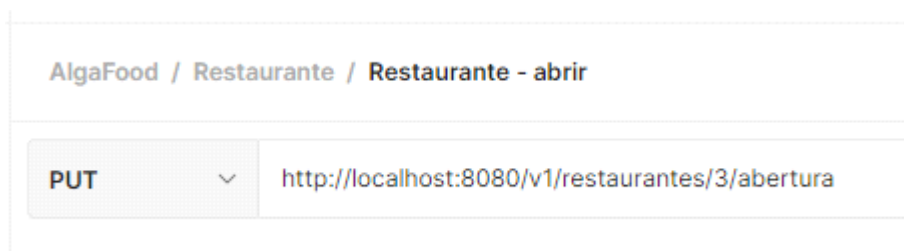
essa anotação ficou da seguinte forma:

```

2 usages
@PreAuthorize("hasAuthority('SCOPE_WRITE') and "
    + "( hasAuthority('EDITAR_RESTAURANTES') or "
    + "@algaSecurity.gerenciaRestaurante(#restauranteId) )")
@Retention(RUNTIME)
@Target(METHOD)
public @interface PodeGerenciarFuncionamento { }

```

estamos capturando o id do restaurante atual, vindo da requisição, no caso id 3



O método "gerenciaRestaurante" faz a lógica de verificação de associação do usuário responsável pelo restaurante da requisição (restaurante 3)

```

1 usage
public boolean gerenciaRestaurante(Long restauranteId){

    return restauranteRepository.existsResponsavel(restauranteId, getIdUsuario());
}

```

getIdUsuario

```

@Component
public class AlgaSecurity {

    1 usage
    @Autowired
    private RestauranteRepository restauranteRepository;

    1 usage
    public Authentication getAuthentication(){
        return SecurityContextHolder.getContext().getAuthentication();
    }

    public Long getIdUsuario() {

        final Jwt jwt = (Jwt) getAuthentication().getPrincipal();

        return jwt.getClaim("usuario_id");
    }
}

```

captura o id do usuário atual do contexto de autenticação, ou seja, o usuário, o usuário 5 (manoel.loja@gmail.com)

AlgaFood - Authorization Server / Password Flow - 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"/>	username	manoel.loja@gmail.com
<input checked="" type="checkbox"/>	password	123
<input checked="" type="checkbox"/>	grant_type	password
	Key	Value

com isso, o usuário 5 só pode gerenciar (abrir e fechar) restaurantes de id 1 e 3

23.29. Restringindo acessos com @PostAuthorize

segunda-feira, 15 de maio de 2023 13:09

Para buscar um pedido pelo seu id, podemos fazer da seguinte maneira

```
@CheckSecurity.Pedidos.PodeBuscar
@Override
@GetMapping("/{codigoId}")
public PedidoDTO buscar(@PathVariable String codigoId){
    return pAssembler.toModel(pedidoService.buscar(codigoId));
}
```

```
1 usage
public @interface Pedidos {

    1 usage
    @PreAuthorize("hasAuthority('SCOPE_READ') and " +
        "( hasAuthority('CONSULTAR_PEDIDOS') or " +
        "@algaSecurity.clienteDoPedido(#codigoId) or " +
        "@algaSecurity.gerenciaRestauranteDoPedido(#codigoId))")
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodeBuscar { }
```

```
1 usage
public boolean clienteDoPedido(String codigoId) {

    return true;
}

1 usage
public boolean gerenciaRestauranteDoPedido(String codigoId){

    return true;
}
```

Podemos ter o seguinte contexto:
O escopo do cliente precisa estar definido como somente leitura para acessar o endpoint de buscar pedidos E (precisa da permissão de consulta de pedidos OU precisa ser o cliente daquele pedido específico OU precisa gerenciar aquele restaurante de onde o pedido foi emitido, utilizando a mesma lógica da aula 23.28.

Mas vamos utilizar a anotação @PostAuthorize

```
1 usage
public @interface Pedidos {

    1 usage
    @PreAuthorize("hasAuthority('SCOPE_READ') and isAuthenticated()")
    @PostAuthorize("hasAuthority('CONSULTAR_PEDIDOS') or "
        + "@algaSecurity.getUsuarioId() == responseObject.cliente.id or "
        + "@algaSecurity.gerenciaRestaurante(responseObject.restaurante.id)")
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodeBuscar { }
```

O PostAuthorize faz a verificação após o processamento do método mas antes dele ser serializado.

a flag "responseObject" retorna a instância do objeto de retorno do método, e com isso, é capaz de acessar o pedido buscado do banco de dados e fazer as validações a partir das informações processadas pelo método. Logo, está retornando a instância do pedido que foi passado por id para ser buscado.

Métodos clienteDoPedido e gerenciaRestauranteDoPedido foram deletados, pois basta reutilizar os métodos anteriores.

```
public Long getUsuarioId() {

    final Jwt jwt = (Jwt) getAuthentication().getPrincipal();

    return jwt.getClaim("usuario_id");
}

2 usages
public boolean gerenciaRestaurante(Long restauranteId){

    return restauranteRepository.existsResponsavel(restauranteId, getUsuarioId());
}
```

O token tem que estar com o escopo READ do cliente, além de ter a permissão de consultar pedidos ou ser dono do pedido ou gerenciar o restaurante do qual o pedido foi emitido

NOTA IMPORTANTE: bug corrigido na aula [23.38. Corrigindo lógica de restrição de acessos para Client Credentials Flow](#) pois getUsuarioId() == responseObject.client.id pode ser verdadeiro caso ambos sejam nulos

23.30. Desafio: restringindo acessos ao endpoint de pesquisa de pedidos

segunda-feira, 15 de maio de 2023 15:38

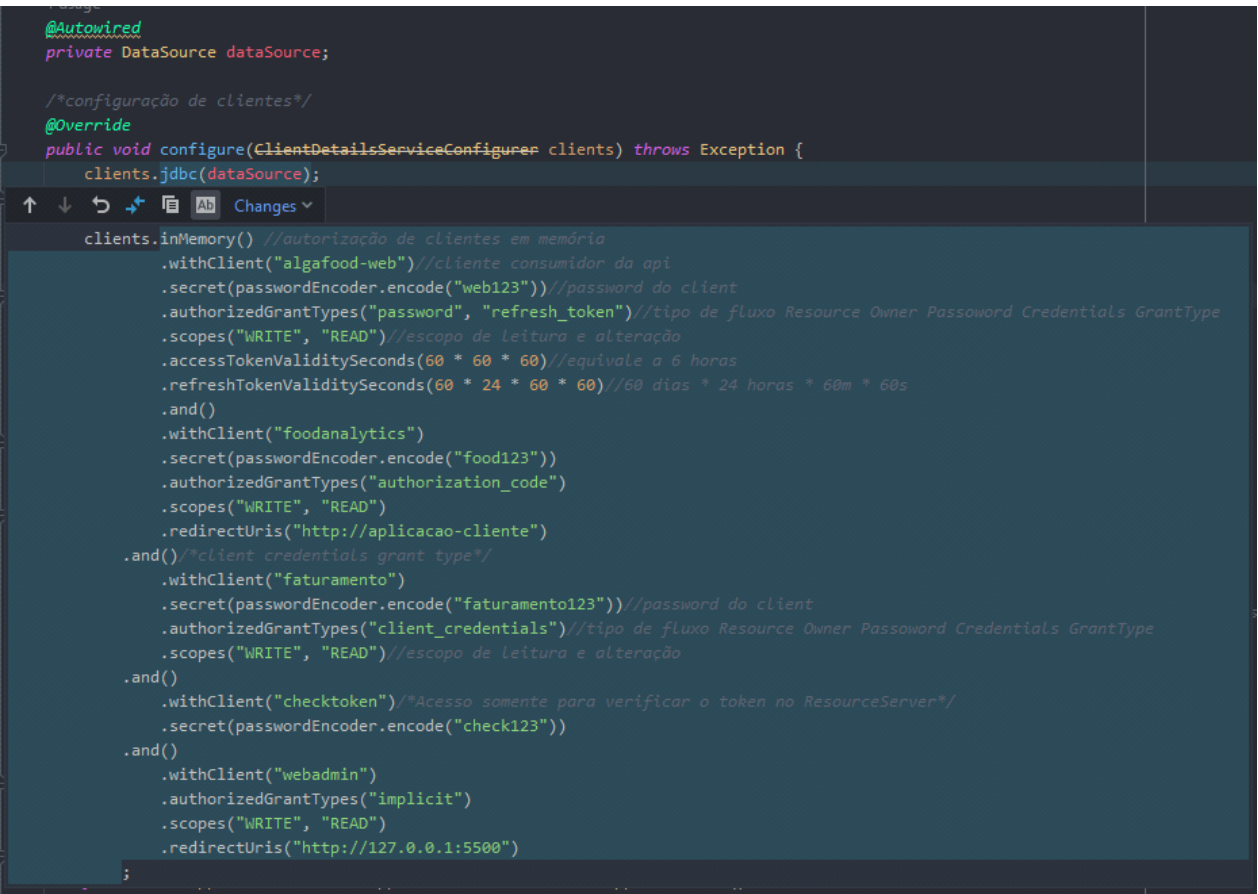
23.31. Desafio: restringindo acessos aos endpoints de transição de status de pedidos

segunda-feira, 15 de maio de 2023 15:38

23.36. Configurando os clientes OAuth2 em um banco de dados SQL

segunda-feira, 15 de maio de 2023 16:57

Estávamos configurando os clientes do A.S em memória, agora refatoramos para utilizar os registros do banco de dados



deletamos todo o trecho de código depois de "inMemory" e adicionamos jdbc(dataSource).

Após isso é necessário criar uma tabela que seja exatamente a da implementação, chamada oauth_client_details

```
create table oauth_client_details (
  client_id varchar(255),
  resource_ids varchar(256),
  client_secret varchar(256),
  scope varchar(256),
  authorized_grant_types varchar(256),
  web_server_redirect_uri varchar(256),
  authorities varchar(256),
  access_token_validity integer,
  refresh_token_validity integer,
  additional_information varchar(4096),
  autoapprove varchar(256),

  primary key (client_id)
) engine=innodb default charset=utf8;
```

23.37. Cadastrando clientes OAuth2 no banco de dados e testando a emissão de tokens

segunda-feira, 15 de maio de 2023 17:09

- Adicionando dados na tabela via afterMigrate

```
#23.37. Cadastrando clientes OAuth2 no banco de dados e testando a emissão de tokens
insert into oauth_client_details (
  client_id, resource_ids, client_secret,
  scope, authorized_grant_types, web_server_redirect_uri, authorities,
  access_token_validity, refresh_token_validity, autoapprove
)
values (
  'algafood-web', null, '$2y$12$w3igMjsfS5XoAYuowoH3C.54vRFWlcXSHLjX7MwF990Kc2KKKh72e',
  'READ,WRITE', 'password', null, null,
  60 * 60 * 6, 60 * 24 * 60 * 60, null
);

insert into oauth_client_details (
  client_id, resource_ids, client_secret,
  scope, authorized_grant_types, web_server_redirect_uri, authorities,
  access_token_validity, refresh_token_validity, autoapprove
)
values (
  'foodanalytics', null, '$2a$12$InuKivuh0ARkBULDgXefP.LEBQhNP1U2RnNDU1Ta1AQQE7ZgygHqy',
  'READ,WRITE', 'authorization_code', 'http://localhost:8082', null,
  null, null, null
);

insert into oauth_client_details (
  client_id, resource_ids, client_secret,
  scope, authorized_grant_types, web_server_redirect_uri, authorities,
  access_token_validity, refresh_token_validity, autoapprove
)
values (
  'faturamento', null, '$2y$12$fHixriC7yXX/i1/CmpnGH.RFyK/15YapLCFOEbIktONjE8ZDykSnu',
  'READ,WRITE', 'client_credentials', null, 'CONSULTAR_PEDIDOS,GERAR_RELATORIOS',
  null, null, null
);
```

23.38. Corrigindo lógica de restrição de acessos para Client Credentials Flow

segunda-feira, 15 de maio de 2023 19:51

```
6 usages
public @interface Pedidos {

    1 usage
    @PreAuthorize("hasAuthority('SCOPE_READ') and isAuthenticated()")
    @PostAuthorize("hasAuthority('CONSULTAR_PEDIDOS') or "
        + "@algaSecurity.usuarioAutenticadoIgual(returnObject.cliente.id) or "
        + "@algaSecurity.gerenciaRestaurante(returnObject.restaurante.id)")
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodeBuscar { }

    1 usage
    @PreAuthorize("hasAuthority('SCOPE_READ') and (hasAuthority('CONSULTAR_PEDIDOS') or "
        + "@algaSecurity.usuarioAutenticadoIgual(#pedidoFilter.clienteId) or "
        + "@algaSecurity.gerenciaRestaurante(#pedidoFilter.restauranteId)")
    @Retention(RUNTIME)
    @Target(METHOD)
    public @interface PodePesquisar { }
```

```
4 usages
public boolean usuarioAutenticadoIgual(Long usuarioId){
    return getUsuarioId() != null && usuarioId != null
        && getUsuarioId().equals(usuarioId);
}
```

23.29

23.39. Gerando links do HAL dinamicamente de acordo com permissões do usuário

segunda-feira, 15 de maio de 2023 20:56

Exibir links do HyperMedia (capítulo de HATEOAS) de respostas baseado no usuário autenticado da requisição.

```
1 usage
@Autowired
private AlgaSecurity algaSecurity;

public PedidoAssembler() { super(PedidoController.class, PedidoDTO.class); }

@Override
public PedidoDTO toModel(Pedido pedido){
    final PedidoDTO pedidoDTO = createModelWithId(pedido.getId(), pedido);
    modelMapper.map(pedido, pedidoDTO);

    if(algaSecurity.podeGerenciarPedidos(pedido.getCodigo())){
        if(pedido.podeSerConfirmado())
            pedidoDTO.add(algaLinks.linkToConfirmacaoPedido(pedido.getCodigo(), rel: "confirmar"));

        if(pedido.podeSerCancelado())
            pedidoDTO.add(algaLinks.linkToCancelamentoPedido(pedido.getCodigo(), rel: "cancelar"));

        if(pedido.podeSerEntregue())
            pedidoDTO.add(algaLinks.linkToEntregaPedido(pedido.getCodigo(), rel: "entregar"));
    }
}
```

23.40. Desafio: gerando links do HAL dinamicamente de acordo com permissões

segunda-feira, 15 de maio de 2023 21:49

23.41. Juntando o Resource Server com o Authorization Server no mesmo projeto

segunda-feira, 15 de maio de 2023 22:59

É preferível que fique tudo na mesma instância pois, para baratear custos de instância de aplicações.

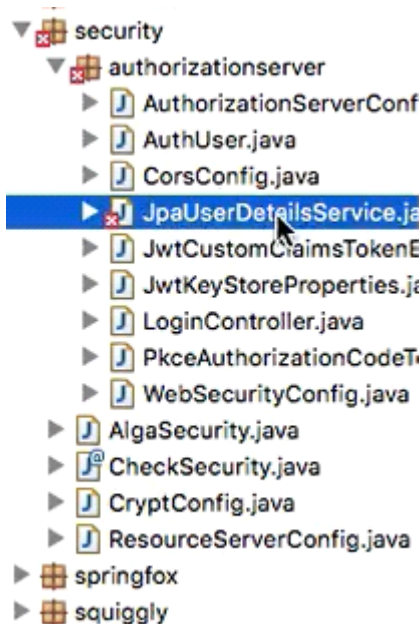
- Copiar as propriedades do pom.xml do jwt e oauth2

```
<!-- 23.41. Juntando o Resource Server com o Authorization Server no mesmo projeto -->
<!-- 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>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-jwt</artifactId>
  <version>1.1.1.RELEASE</version>
</dependency>
```

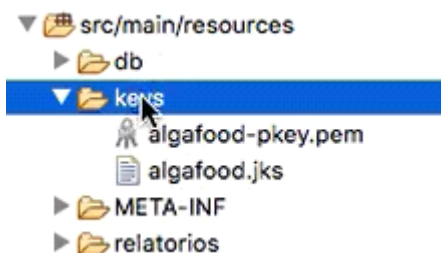
oauth2: stack antiga para o authorization server

- Copiar arquivos necessário para a configuração do Auth. Server





- copiar o .jks para a aplicação



- copiar propriedades

```

9
10 algafood.jwt.keystore.path=keystores/algafood.jks
11 algafood.jwt.keystore.password=123456
12 algafood.jwt.keystore.keypair-alias=algafood
13

```

Nota: alterar o caminho para o algafood.jks

Copiar a forma que capturamos o arquivos do classpath do keys/algafood.jks para igual ao classpath:keys/algafood-pkey.pem

```

spring.security.oauth2.resourceserver.jwt.public-key-location=classpath:keys/algafood-pkey.pem
algafood.jwt.keystore.path=keys/algafood.jks
algafood.jwt.keystore.password=123456
algafood.jwt.keystore.keypair-alias=algafood

```

Isso ocorre porque a variável public-key-location é do tipo Resource

```

/**
 * Location of the file containing the public key used to verify a JWT.
 */
private Resource publicKeyLocation;

public String getJwkSetUri() {
    return this.jwkSetUri;
}

```

logo, basta apenas declarar a propriedade como Resource

```

@NotBlank
private Resource path;

@NotBlank

@Setter
@Getter
@Validated
@Component
@ConfigurationProperties("algafood.jwt.keystore")
public class JwtKeyStoreProperties {

    @NotNull
    private Resource jksLocation;
}

```

```

spring.security.oauth2.resourceserver.jwt.public-key-location=classpath:keys/
algafood.jwt.keystore.jks-location=classpath:keys/algafood.jks
algafood.jwt.keystore.password=123456

```

```

properties application.properties JwtKeyStoreProperties.java AuthorizationServerConfig.java
@Bean
public JwtAccessTokenConverter jwtAccessTokenConverter() {
    var jwtAccessTokenConverter = new JwtAccessTokenConverter();

    var keyStorePass = jwtKeyStoreProperties.getPassword();
    var keyPairAlias = jwtKeyStoreProperties.getKeypairAlias();

    var keyStoreKeyFactory = new KeyStoreKeyFactory(
        jwtKeyStoreProperties.getJksLocation(), keyStorePass.toCharArray());
    var keyPair = keyStoreKeyFactory.getKeyPair(keyPairAlias);

    jwtAccessTokenConverter.setKeyPair(keyPair);

    return jwtAccessTokenConverter;
}

```

O A.S e o R.S tinham configurações para o CORS definidas individualmente, porém, agora que os dois projetos foram juntados, temos que eliminar um dos dois.

A CorsConfig do A.S. O padrão foi definido para todas as requisições e não somente em oauth/token

```

package com.algaworks.algafood.core.security; // Fonte: https://spring.io/blog/2015/06/08/cors-sur

import ...

@Configuration
public class CorsConfig {

    @Bean
    public FilterRegistrationBean<CorsFilter> filterRegistrationBean() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(false);

        config.setAllowedOriginPatterns(Collections.singletonList("http://**"));
        config.setAllowedMethods(Collections.singletonList("*"));
        config.setAllowedHeaders(Collections.singletonList("*"));

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( pattern: "**", config); // cors para todas as requisições

        FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<>();
        bean.setFilter(new CorsFilter(source));
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);

        return bean;
    }
}

```

e a do R.S

```

package com.algaworks.algafood.core.web;

import ...

@Configuration
public class WebConfig implements WebMvcConfigurer {

    // usage
    @Autowired
    ApiRetirementHandler apiRetirementHandler;

    // foi retirado pois na classe CorsConfig, a configuração de lá já surte o mesmo efeito
    // @Override
    // public void addCorsMappings(CorsRegistry registry) {
    //     registry.addMapping("/**") // todos os endpoints
    //         .allowedMethods("*"); // todos os métodos http
    // }

    @Bean
    public Filter shallowEtagHeaderFilter() { return new ShallowEtagHeaderFilter(); }

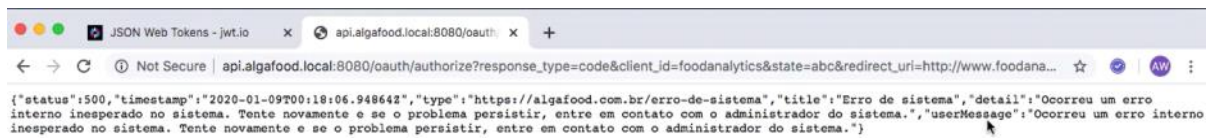
    @Override
    public void addInterceptors(InterceptorRegistry registry) { registry.addInterceptor(apiRetirementHandler); }
}

```

foi retirado pois na classe CorsConfig, a configuração de lá surte o mesmo efeito.

A do A.S é necessário para as emissões de token, porém, surtem efeito em toda a aplicação.

Depois dos ajustes, podemos rodar o projeto, mas pode dá esse erro



Adicionar no método

`.authorizeRequests().andMatchers`

Quando uma requisição iniciar nesse padrão, ela será redirecionada para a página de login

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/oauth/**").authenticated()
            .and()
            .csrf().disable()
    }
}
```

adicionar o form login

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .formLogin()
        .and()
        .authorizeRequests()
            .antMatchers("/oauth/**").authenticated()
    }
}
```

23.42. Ajustando a documentação da API para suporte a OAuth2

segunda-feira, 15 de maio de 2023 23:45

23.43. Customizando a página de login

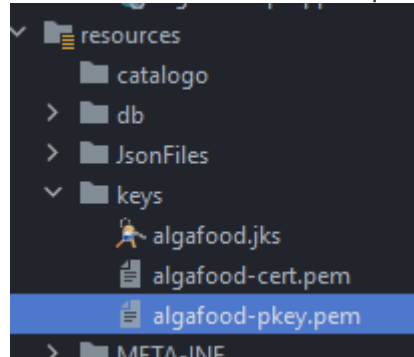
segunda-feira, 15 de maio de 2023 23:54

23.45. Implementando o endpoint do JSON Web Key Set (JWKS)

terça-feira, 16 de maio de 2023

11:06

Temos um arquivo `algafood-pkey.pem` que representa uma chave pública, para que o resource server consiga validar a confiabilidade daquele token recebido, ou legítimo.



a chave pública está dentro do arquivo `.pem` e a privada está dentro do `.jks`.

```
#23.12. Configurando a validação de JWT no Resource Server com a chave pública
spring.security.oauth2.resourceserver.jwt.public-key-location=classpath:keys/algafood-pkey.pem
#em vez de um arquivo, o R.S vai buscar no endpoint a key publica para fazer a validação do token
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=http://localhost:8080/oauth2/jwks

#####23.41. Juntando o Resource Server com o Authorization Server no mesmo projeto#####

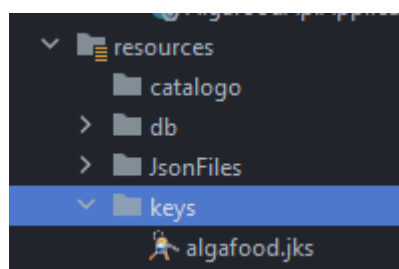
#configurações vindas do outro projeto Algafood-auth

#22.8. Criando o projeto do Authorization Server com Spring Security OAuth2
server.port=8081

#23.2. Configurando o RedisTokenStore
#necessário armazenar token pois vamos usar token transparente
spring.redis.host=localhost
spring.redis.password=
spring.redis.port=6379

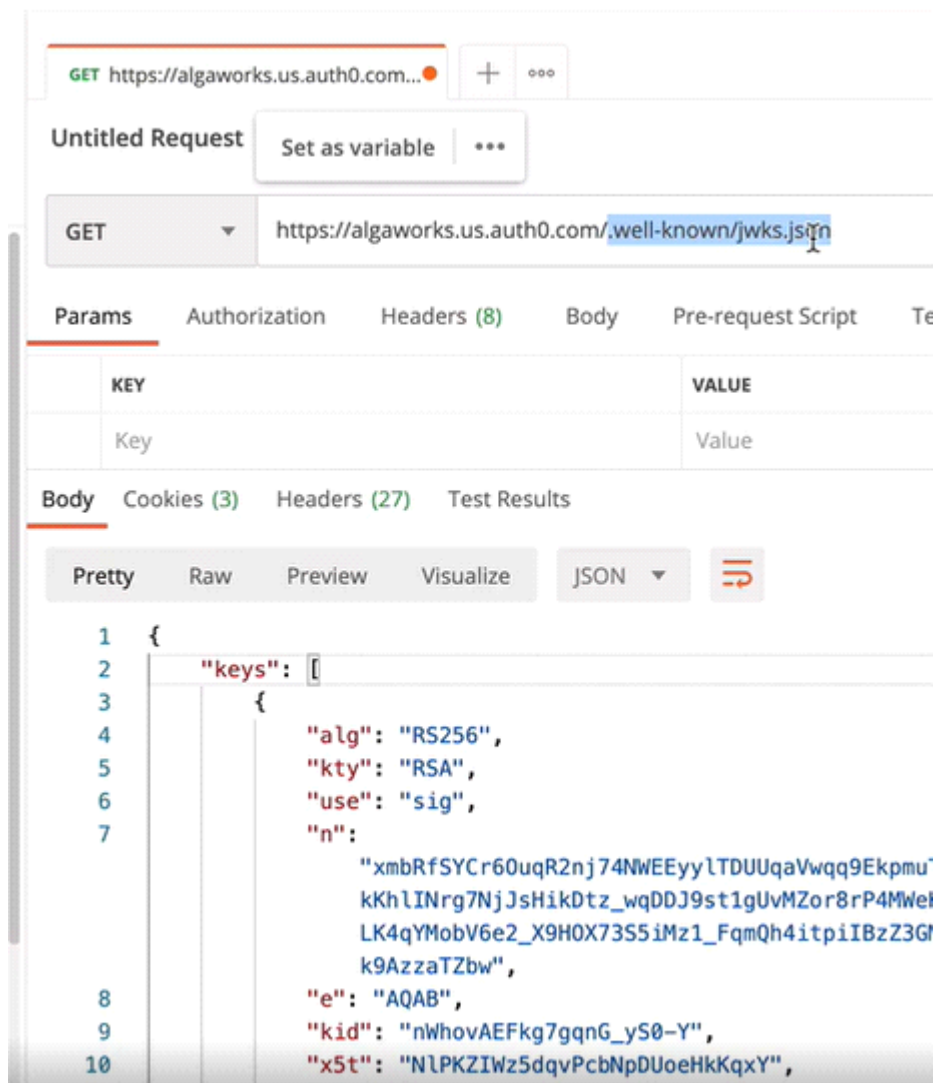
#23.5. Gerando JWT com chave simétrica (HMAC SHA-256) no Authorization Server
#23.10. Desafio: criando bean de propriedades de configuração do KeyStore
algafood.jwt.keystore.jks-location=classpath:keys/algafood.jks
```

- Para não expor, precisamos deletar os arquivos



JWKS é um conjunto de de chaves contendo as chaves públicas para verificar as JWT emitidos pelo Authorization Server (assinados pelo algoritmo sha256)

vamos expor as chaves públicas por um endpoint



auth0 é um servidor que comporta várias chaves.

Quem tem uma chave pública pode somente validar o token, quanto a privada vai emitir o token.

Agora precisamos implementar um endpoint JWK no Authorization Server.

- Precisamos do keypar e um método que retorna uma instância de JWK

```

@Bean
public JWKS jwkSet(){
    final KeyPair keyPair = getKeyPair();

    |
}

2 usages
@Bean/*Converte informações de user logado para JWT * pode ser usado como Bean*/
public JwtAccessTokenConverter jwtAccessTokenConverter (){
    /*utiliza hmacsha-256 simétrico*/
    JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
    KeyPair keyPair = getKeyPair();
    jwtAccessTokenConverter.setKeyPair(keyPair);
    return jwtAccessTokenConverter;
}

2 usages
private KeyPair getKeyPair(){
    //jwtAccessTokenConverter.setSigningKey("ansbchd978234dkfjhsd98sd42nlkj2094kwejs0d8g");

    String keyStorePass = jwtKeyStoreProperties.getPassword();
    String keyPairAlias = jwtKeyStoreProperties.getKeypairAlias();

    final KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(jwtKeyStoreProperties.getJksLocation(),
    final KeyPair keyPair = keyStoreKeyFactory.getKeyPair(keyPairAlias);

    return keyPair;
}

```

foi criado um método utilitário para pegar instâncias de keypar.

```

@Bean/*23.45. Implementando o endpoint do JSON Web Key Set (JWKS)*/
public JWKS jwkSet(){
    final KeyPair keyPair = getKeyPair();

    /*pegando a chave pública do keypair*/
    RSAKey rsaKey = new RSAKey.Builder((RSAPublicKey)keyPair.getPublic())
        .keyUse(KeyUse.SIGNATURE)//chave assinada por nós
        .algorithm(JWSAlgorithm.RS256)
        .keyID(kid: "algafood-key-id").build();//pode ser qualquer nome de iden.

    return new JWKS(rsaKey);
}

2 usages
@Bean/*Converte informações de user logado para JWT * pode ser usado como Bean*/
public JwtAccessTokenConverter jwtAccessTokenConverter (){
    /*utiliza hmacsha-256 simétrico*/
    JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
    KeyPair keyPair = getKeyPair();
    jwtAccessTokenConverter.setKeyPair(keyPair);
    return jwtAccessTokenConverter;
}

2 usages
private KeyPair getKeyPair(){
    //jwtAccessTokenConverter.setSigningKey("ansbchd978234dkfjhsd98sd42nlkj2094kwejsd")

    String keyStorePass = jwtKeyStoreProperties.getPassword();
    String keyPairAlias = jwtKeyStoreProperties.getKeypairAlias();

    final KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(jwtKeyStoreProperties.getKeyStorePath(), keyStorePass.toCharArray());
    final KeyPair keyPair = keyStoreKeyFactory.getKeyPair(keyPairAlias);

    return keyPair;
}

```

o JWK foi configurado, agora precisamos expor um endpoint que dê acesso ao JWK

```

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

import ...

@RestController
public class KwkSetController {

    1 usage
    @Autowired
    private JWKS jwkSet;

    @GetMapping("/.well-know/jwks.json")
    public Map<String, Object> keys (){

        return this.jwkSet.toJSONObject();
    }
}

```

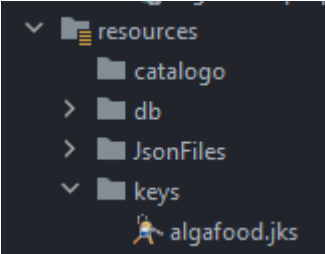
agora precisamos substituir o classpath que trazia o public key local para um remoto

```
#23.45. Implementando o endpoint do JSON Web Key Set (JWKS)
#a linha 108 foi comentada para a nova configuração de location para set uri
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=http://localhost:8080/.well-know/jwks.json
```

Nota: o endpoint é chamado apenas algumas vezes na hora de fazer a validação e não todas as vezes

23.46. Externalizando o KeyStore: criando um ProtocolResolver para Base64

terça-feira, 16 de maio de 2023 14:41



java key stores:
par de chaves {publica e privada} para validação e emissão de tokens jwt

Temos uma senha para extrair as informações, junto com o alias do conjunto.

```
algafood.jwt.keystore.jks-location=classpath:keys/algafood.jks
algafood.jwt.keystore.password=123456
algafood.jwt.keystore.keypair-alias=algafood
```

Mas vamos retirar do projeto esse arquivo pois está exposto. Iremos carregar esse arquivo externamente.

vamos implementar um novo protocolo (além de classpath, file, http e https) para carregar um resource em base64.