



ENSINO MÉDIO INTEGRADO - INFORMÁTICA

Disciplina de Desenvolvimento Web | Backend

[Aula 10](#): Framework Spring Boot - Autenticação e Autorização

Gil Eduardo de Andrade

Conceitos Preliminares

(<https://docs.spring.io/spring-boot/documentation.html>)

(<https://www.jhipster.tech/creating-an-app/>)

Autenticação e Autorização

A autenticação e autorização no Spring Boot com *JHipster* são baseadas no *Spring Security*, um poderoso framework do ecossistema Java focado em fornecer autenticação e autorização altamente customizáveis.

Spring Security

O *Spring Security* é o padrão oficial para proteger aplicações e APIs corporativas, atuando como uma cadeia de filtros que intercepta requisições web antes que cheguem à lógica de negócio

Como Funciona?

O *Spring Security* atua como um sistema de segurança completo e extensível. Os dois pilares fundamentais da biblioteca são:

- Autenticação: processo de verificar a identidade do usuário (ex: login e senha, tokens JWT, ou OAuth2).
- Autorização: processo de determinar o que um usuário autenticado tem permissão para fazer ou acessar (ex: controle de roles ou permissões de administrador).

Ele lida nativamente com dezenas de boas práticas e proteções, prevenindo ataques comuns como sequestro de sessão (CSRF) e roubo de dados, poupando você de ter que programar sistemas complexos de segurança do zero

Mecanismos de Autenticação

- JWT (Padrão para Microsserviços e muito usado em Monolitos): é uma abordagem stateless. O usuário faz login enviando suas credenciais, e o servidor responde com um token criptografado que contém as informações

de quem é o usuário e suas permissões. Esse token é enviado no cabeçalho Authorization a cada requisição subsequente.

- OAuth 2.0 / OIDC (Padrão do JHipster para (*)Single Sign-On - SSO): delega a autenticação para um servidor de identidade de terceiros (como Keycloak, Okta ou Auth0). Sua aplicação atua apenas como um Resource Server que valida o token fornecido pelo provedor externo.

()Single Sign-On é o processo de autenticar-se uma única vez para acessar vários aplicativos sem precisar repetir o login. Um exemplo clássico é utilizar a conta do Google ou da Microsoft para entrar em outras páginas ou aplicativos automaticamente.*

Controle de Autorização

O *JHipster* configura automaticamente classes de mapeamento para o controle de acesso baseado em funções (RBAC) através de duas roles padrão: ROLE_USER e ROLE_ADMIN.

No Backend (Spring Security): a autorização é tratada no lado do servidor utilizando as anotações do próprio Spring Boot, como @Secured, @PreAuthorize e @PostAuthorize nas suas classes de serviço ou controladores REST.

Exemplo:

```
Java
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
public void removeUser(Long id) { ... }
```

No Frontend (Angular/React/Vue/Svelte): O *JHipster* também cuida da autorização do lado do cliente. Os arquivos de rotas utilizam um "*User Route Access Service*" para limitar quais telas o usuário consegue acessar, ocultando componentes ou redirecionando requisições caso ele não possua o *token* ou as *roles* corretas.

Funcionamento do Ciclo de Requisição

- Login: o usuário envia usuário e senha para o endpoint de autenticação (ex: /api/authenticate).
- Validação: o *AuthenticationManager* do Spring Security checa as credenciais no banco de dados da aplicação ou as encaminha para o servidor de identidade (OIDC).

- Emissão de Token: O servidor emite o token (JWT ou access token do provedor OIDC) e o devolve ao cliente.
- Requisições de Acesso: o cliente anexa este token no cabeçalho HTTP. A cadeia de filtros (*Filter Chain*) intercepta a chamada, valida a assinatura e expiração do token, e carrega o contexto do usuário autenticado para dentro da sessão atual da requisição.

Vantagens e Recursos

- Proteção automática: ao adicionar a dependência do *Spring Security*, todas as rotas da sua aplicação são bloqueadas e passam a exigir autenticação por padrão.
- Altamente customizável: permite criar lógicas de segurança em memória, com banco de dados relacional (JPA/JDBC), ou integrar com servidores de identidade externos (como Keycloak, Okta, ou Auth0).
- Suporte a padrões modernos: integração nativa para emitir e validar JSON Web Tokens (JWT) e OAuth2, fundamentais para APIs REST e microserviços modernos.

IMPLEMENTANDO: AUTENTICAÇÃO E AUTORIZAÇÃO

Continuação da Aplicação Criada na Aula Anterior

1) Definir as Novas Autoridades (Roles)

O *JHipster* já possui, por padrão, os papéis *ROLE_ADMIN* e *ROLE_USER*. Contudo, como queremos trabalhar com os papéis de professor e coordenador, torna-se necessário adicionar *ROLE_PROFESSOR* e *ROLE_COORDENADOR*.

Codificação:

Vamos adicionar as novas constantes para os papéis de professor e coordenador.

Alterar: "`src/main/java/com/aula/springapp/security/AuthoritiesConstants.java`"

Arquivo: "`src/main/java/com/aula/springapp/security/AuthoritiesConstants.java`"

Java

```
public final class AuthoritiesConstants {  
    public static final String ADMIN = "ROLE_ADMIN";
```



```
public static final String USER = "ROLE_USER";  
public static final String ANONYMOUS = "ROLE_ANONYMOUS";  
// Linhas Adicionadas  
public static final String PROFESSOR = "ROLE_PROFESSOR";  
public static final String COORDENADOR = "ROLE_COORDENADOR";  
// ...  
}
```

Banco de Dados (MongoDB):

Como o *JHipster* gerencia as autoridades no banco, você precisa inserir esses novos papéis na coleção *authority*.

Sendo assim, vamos acessar o Mongo Express (<http://localhost:18001>) e inserir dois novos documentos na coleção *authority*:

```
Java  
{  
  _id: 'ROLE_PROFESSOR',  
  _class: 'com.aula.springapp.domain.Authority'  
}
```

OBS.: crud ⇒ jhi_authority ⇒ New Document

```
Java  
{  
  _id: 'ROLE_COORDENADOR',  
  _class: 'com.aula.springapp.domain.Authority'  
}
```

OBS.: crud ⇒ jhi_authority ⇒ New Document

2) Configurando a Segurança Global

Primeiramente vamos considerar, para aplicação em questão, e antes de efetuarmos as codificações necessárias, quais são as permissões que cada papel terá na aplicação. Neste contexto, a tabela a seguir especifica as permissões que os papéis de professor e coordenador possuem para manipular Cursos e Disciplinas.

PROFESSOR - PERMISSÕES		
	CURSO	DISCIPLINA
CRIAR	Não	Sim



LISTAR	Sim	Sim
BUSCAR	Sim	Sim
ALTERAR	Não	Sim
REMOVER	Não	Sim
COORDENADOR - PERMISSÕES		
	CURSO	DISCIPLINA
CRIAR	Sim	Não
LISTAR	Sim	Sim
BUSCAR	Sim	Sim
ALTERAR	Sim	Não
REMOVER	Sim	Não

Com as permissões definidas, podemos restringir as rotas no arquivo:

“src/main/java/com/aula/springapp/config/SecurityConfiguration.java”

Sendo assim, vamos substituir as linhas que havíamos liberado na aula anterior, utilizando ***“.permitAll()”*** pelas novas regras de autoridade:

```
Java
.authorizeHttpRequests(authz ->
    authz
        // ... outras rotas permitAll ...

        // Regras para CURSOS
        .requestMatchers(HttpMethod.GET, "/api/cursos/**")
        .hasAnyAuthority(AuthoritiesConstants.COORDENADOR,
            AuthoritiesConstants.PROFESSOR)

        .requestMatchers("/api/cursos/**")
        .hasAuthority(AuthoritiesConstants.COORDENADOR)

        // Regras para DISCIPLINAS
        .requestMatchers(HttpMethod.GET, "/api/disciplinas/**")
        .hasAnyAuthority(AuthoritiesConstants.PROFESSOR,
            AuthoritiesConstants.COORDENADOR)
```



```
.requestMatchers("/api/disciplinas/**")  
.hasAuthority(AuthoritiesConstants.PROFESSOR)  
  
.requestMatchers("/api/**").authenticated()  
// ...  
)
```

Compreendendo a Codificação

- Método GET: tanto Professor quanto Coordenador podem listar/buscar (Cursos e Disciplinas).
- Métodos POST, PUT, DELETE: apenas o Coordenador pode alterar Cursos, e apenas o Professor pode alterar Disciplinas.

3) Criando Usuários de Teste

Para que seja possível testar o processo de autenticação e autorização é necessário termos usuários vinculados aos papéis definidos.

Há duas possibilidades, criar os usuários via API (se a rota /api/register estiver aberta) ou diretamente no MongoDB na coleção user.

Dica: o *JHipster* salva as senhas com Hash (BCrypt). Sendo assim, é mais fácil duplicar o usuário admin no Mongo Express e apenas alterar o login e o campo authorities para ["ROLE_PROFESSOR"] ou ["ROLE_COORDENADOR"]

Subindo a Aplicação

No Terminal de Comandos
(Dentro da pasta da aplicação)

Shell

```
docker compose up
```

Autenticando como Administrador

No Terminal de Comandos
Autenticação - Administrador (POST)



Shell

```
curl -X POST "http://localhost:18000/api/authenticate" \
-H "Content-Type: application/json" \
-d '{"username": "admin", "password": "admin"}'
```

```
% curl -X POST "http://localhost:18000/api/authenticate" \
-H "Content-Type: application/json" \
-d '{"username": "admin", "password": "admin"}'
{"id_token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbGlzImV4cC
CIsIm1hdCI6MTc4NTI2NDZ0CwidXNlcklkIjoidXNlci0xIn0.9-2TVJ9-2
pQ"}%
```

Criando o Usuário Professor

No Terminal de Comandos

Criando Usuário - Professor (POST)

Shell

```
curl -X POST "http://localhost:18000/api/admin/users" \
-H "Authorization: Bearer $ADMIN_TOKEN" \
-H "Content-Type: application/json" \
-d '{
  "login": "professor",
  "firstName": "Professor",
  "lastName": "Silva",
  "email": "professor@aula.com",
  "activated": true,
  "langKey": "pt-br",
  "authorities": ["ROLE_USER", "ROLE_PROFESSOR"]
}'
```

Criando o Usuário Coordenador

No Terminal de Comandos

Criando Usuário - Coordenador (POST)

Shell

```
curl -X POST "http://localhost:18000/api/admin/users" \
-H "Authorization: Bearer $ADMIN_TOKEN" \
-H "Content-Type: application/json" \
-d '{
  "login": "coordenador",
```

```
"firstName": "Coordenador",  
"lastName": "Souza",  
"email": "coordenador@aula.com",  
"activated": true,  
"langKey": "pt-br",  
"authorities": ["ROLE_USER", "ROLE_COORDENADOR"]  
}'
```

Mongo Express Database: crud Collection: jhi_user

 	user-1	admin	\$2a\$10\$gSAhZrxMlrbgi/kkK9UceBPpChGWJA7SYlb1Mqo.n...	admin	Administrator	admin@localhost	true
 	professor		\$2a\$10\$i7J.VkWIwautvkf80bocl.LpZh6m3h9ds19VS7h4Mw...	Professor	Silva	professor@aula.com	true
6a68f4c27decc280e8f7c58d							
 	coordenador		\$2a\$10\$dZO0n1.7DlDdJr/58L1GG0vcxP1kmqxy/o72YEQW0b...	Coordenador	Souza	coordenador@aula.com	true
6a68f5837decc280e8f7c58e							

Importante Sobre a Senha

Ao criar um usuário via **“/api/admin/users”**, o *JHipster* não permite enviar a senha no JSON por questões de segurança. Ele gera uma senha aleatória.

Para definir a senha **“admin”** para esses novos usuários e poder efetuar os testes posteriormente:

- Abra o Mongo Express.
- Abra a coleção **“user”**.
- Edite os usuários professor e coordenador.
- Copie o valor do campo password do usuário admin (que é o hash para a senha **“admin”**) e cole nos novos usuários.

Pode ser Necessário

O *JHipster* utiliza cache para usuários para melhorar a performance. Pode ser que o Spring ainda esteja **“lembrando”** do usuário antes de efetuarmos a alteração da senha no banco. Sendo assim, pode ser necessário reiniciar a aplicação para limpar o cache.



No Terminal de Comandos
Autenticação - Professor (POST)

Shell

```
docker compose restart app
```

4) Testando a Aplicação

No Terminal de Comandos
Autenticação - Professor (POST)

Shell

```
curl -X POST "http://localhost:18000/api/authenticate" \
-H "Content-Type: application/json" \
-d "{\"username\": \"professor\", \"password\": \"admin\"}"
```

No Terminal de Comandos
Autenticação - Coordenador (POST)

Shell

```
curl -X POST "http://localhost:18000/api/authenticate" \
-H "Content-Type: application/json" \
-d "{\"username\": \"coordenador\", \"password\": \"admin\"}"
```

No Terminal de Comandos
Criar Curso - Papel de Professor (POST)

Shell

```
curl -v -X POST "http://localhost:18000/api/cursos" \
-H "Authorization: Bearer TOKEN_PROF" \
-H "Content-Type: application/json" \
-d "{\"nome\": \"Técnico em Produção Cultural\", \"duracao\": 4}"
```

OBS.: Deve retornar 403 Forbidden / Acesso Negado.

No Terminal de Comandos
Criar Curso - Papel de Coordenador (POST)

Shell

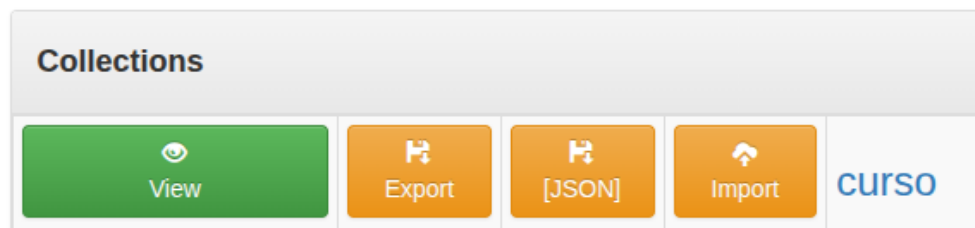
```
curl -v -X POST "http://localhost:18000/api/cursos" \
```



```
-H "Authorization: Bearer TOKEN_COORD" \  
-H "Content-Type: application/json" \  
-d "{\"nome\": \"Técnico em Produção Cultural\", \"duracao\": 4}"
```

OBS.: Deve retornar 201 Created.

Viewing Database: crud



_id	nome	duracao
  6a68fcd247db95356a892615	Técnico em Produção Cultural	4

No Terminal de Comandos

Criar Disciplina - Papel de Professor (POST)

Shell

```
curl -v -X POST "http://localhost:18000/api/disciplinas" \  
-H "Authorization: Bearer TOKEN_PROF" \  
-H "Content-Type: application/json" \  
-d "{\"nome\": \"Artes Cênicas\", \"cargaHoraria\": 40}"
```

OBS.: Deve retornar 201 Created.



Viewing Database: crud

Collections				
 View	 Export	 [JSON]	 Import	curso
 View	 Export	 [JSON]	 Import	disciplina



_id	nome	carga_horaria
  6a68fd5247db95356a892616	Artes Cênicas	40

No Terminal de Comandos

Criar Disciplina - Papel de Coordenador (POST)

Shell

```
curl -v -X POST "http://localhost:18000/api/disciplinas" \  
  -H "Authorization: Bearer TOKEN_COORD" \  
  -H "Content-Type: application/json" \  
  -d "{\"nome\": \"Artes Cênicas\", \"cargaHoraria\": 40}"
```

OBS.: Deve retornar 403 Forbidden / Acesso Negado.