**Spring Boot e Teste: Profiles, Testes e Deploy**

Atualização do Spring

Para atualizar um projeto devemos atualizar a versão no pom.xml

Ter atenção porque talvez existe novas funcionalidades então devemos pesquisar se não irá gerar problemas.

No GitHub do projeto Spring tem todas as versões e também as mudanças de versão para versão.

github.com/spring-projects/spring-boot

Segurança

Recapitulação de segurança

Migramos todo o código para utilizar aquele esquema de autenticação *stateless*. Então usamos a biblioteca JJWT para fazer a geração de Tokens.

Quando o usuário quer acessar o fórum da Alura, quer se autenticar, primeiro ele entra no endereço/auth, aquele *endpoint* para se autenticar; passa o e-mail e a senha. Se tiver os dados corretos nós devolvemos um Token e a aplicação *frontend* dessa pessoa tem que guardar esse Token e nas próximas aquisições sempre enviar esse Token como um cabeçalho.

E tivemos que criar aquela classe que faz toda essa lógica de recuperar esse Token do cabeçalho, validar, forçar uma autenticação durante essa requisição e liberar a requisição para o usuário.

Então fizemos todo esse trabalho utilizando Spring Security e a biblioteca para gerar Tokens JJWT.

Só que em todo esse processo nós focamos basicamente na parte de autenticação. Eu queria restringir algumas URL’s. Na Security Configurations, tem algumas URL’s que estão públicas para listar os tópicos, para detalhar um tópico, para se autenticar. Mas todo o resto está restrito. Só que a minha preocupação era toda nessa parte de autenticação.

Então antes de executar a lógica, a regra de negócio, eu precisava verificar se o usuário estava logado, se ele tinha o Token, se estava autentificado. E deixamos de fora a parte de autorização.

**Autorização baseada em Roles**

E nem sempre numa aplicação basta a pessoa estar logada. Às vezes têm regras de negócio que envolvem perfil de acesso. Você está autentificado, mas você tem permissão para executar essa ação no sistema? Então foi isso que ficou faltando e é justamente.

Vamos imaginar que a regra não é desse jeito, vamos pensar que a regra vai funcionar da seguinte maneira: só pode excluir um tópico se a pessoa tiver o perfil de moderador.

Nós fazemos isso direto na classe SecurityConfigurations, no mesmo local onde configuramos as URL’s do projeto.

Eu vou copiar a última linha de URL, vou colar logo abaixo e modificar para: .antMatchers(HttpMethod.DELETE, “/topicos/\*”) . A requisição é DELETE, que configuramos no Controller e é /tópicos/ seguida de alguma coisa, que no caso é o id.

Para restringir o método será o método chamado hasRole. E dentro eu passo qual é o Role, qual é o perfil do usuário.

Então só pode executar uma requisição do tipo de DELETE para o endereço/topicos/\* se tiver o perfil que eu vou colocar o nome. Você pode chamar de qualquer coisa, eu vou chamar de moderador. E pronto: .antMatchers(HttpMethod.DELETE, “/topicos/\*”).hasRole(“MODERADOR”).

Só isso, não precisa fazer mais nada. Então o Controller nem vai ver que tem essa regra. Na verdade, o Spring Security faz esse controle para nós.

Quando chamarmos o endereço @DeleteMapping(“/{id}”), o Spring verifica o perfil da pessoa que está logada. Se for moderador, então ele entra e executa linha por linha. Se não for moderador ele nem entra, ele devolve um 403, um *forbidden* como resposta. Então isso deixa o nosso código do Controller bem limpo.

Qual a maneira correta de restringir o acesso a determinado endpoint, baseado no perfil do usuário?

Adicionando a chamada ao método hasRole(“NOME\_DO\_ROLE”) no código de configuração do endpoint na classe SecurityConfigurations.

Essa é justamente a maneira que aprendemos a restringir o acesso a determinado endpoint, baseado no role do usuário.

Nesta aula, aprendemos que:

* Para atualizar a versão do Spring Boot na aplicação, basta alterar a tag <version> da tag <parent>, no arquivo pom.xml.
* É importante ler as **release notes** das novas versões do Spring Boot, para identificar possíveis quebras de compatibilidades ao atualizar a aplicação.
* É possível restringir o acesso a determinados endpoints da aplicação, de acordo com o perfil do usuário autenticado, utilizando o método hasRole(“NOME\_DO\_ROLE”) nas configurações de segurança da aplicação.

**Spring profiles**

Qual o principal motivo de se utilizar profiles?

Permitir ter múltiplas configurações distintas na aplicação.

Com profiles podemos ter configurações distintas para cada tipo de ambiente, como *desenvolvimento*, *testes* e *produção*.