04

# Beans baseados em profiles

## Transcrição

[00:00] Olá, bem-vindos de volta ao curso de Spring Boot na Alura. Na última aula discutimos um pouco sobre a questão de ambientes, de configurações, funcionalidades distintas em diferentes ambientes. Na aula de hoje vamos utilizar de fato essa questão de *profiles*, como é chamada no Spring.

[00:18] No nosso caso, qual será o cenário de utilização dos *profiles*? Eu estou com um projeto rodando no meu Eclipse e eu quero fazer o seguinte teste com vocês: eu vou entrar no Postman e vou disparar uma requisição GET para aquele *endpoint* `/topicos` , que é o *endpoint* para listar todos os tópicos.

[00:35] Não sei se vocês lembram , mas esse *endpoint* está configurado como público. Então não estou autenticado. E já voltou a resposta, ele trouxe todos os registros, não precisei me autenticar, mandar Token, nada do gênero.

[00:48] Só que se eu tentar disparar a requisição com DELETE para o `/topicos/1` , que foi aquela última funcionalidade em que mexemos, ele responde com 403, "forbidden". Porque eu tentei disparar uma requisição para um *endpoint* que é privado, está restrito. E tínhamos configurado a questão do Role, do perfil, que só um moderador pode excluir um tópico.

[01:12] Desde que colocamos segurança no nosso projeto isso acabou se tornando um negócio chato durante o desenvolvimento. Sempre que queremos testar, simular uma requisição para uma *endpoint* que é restrito, caímos nesse problema do 403.

[01:27] Então precisamos disparar primeiro a requisição para o *endpoint* `/auth` para se autenticar, pegar aquele Token, guarda-lo e nas próximas requisições de levá-lo no cabeçalho "Authorization", e isso acaba sendo um negócio meio chato e atrapalha um pouco na produtividade.

[01:43] No ambiente de produção é dessa maneira que tem que funcionar, tem que ter autenticação, tem que ter autorização.

[01:49] Mas para o ambiente de desenvolvimento, enquanto eu estou escrevendo o código, escrevendo as funcionalidades, seria muito mais simples e produtivo se eu não tivesse essa parte de segurança, se todos os endereços fossem públicos. Eu não preciso me autenticar, não preciso ficar gerando e enviando Token nas requisições.

[02:06] É justamente isso faremos no nosso projeto. Nós vamos desligar as configurações de segurança no ambiente de desenvolvimento.

[02:14] No ambiente de produção eu quero que continue do jeito que está, mas no ambiente de desenvolvimento eu quero desabilitar o Spring Security, digamos assim.

[02:22] Se abrirmos a classe SecurityConfigurations, é justamente onde estão todas as nossas configurações de segurança. E tem também aquele *endpoint* que acabamos de testar; o `DELETE, "/topicos/*"` tem que estar logado e tem que ter o perfil de moderador. Qualquer outra URL que não estiver aqui tem que estar autenticada. E isso é o que torna esse trabalho de teste improdutivo.

[02:48] Como vamos resolver esse problema com *profile*? Eu vou duplicar a classe SecurityConfigurations dando um "Ctrl + C", "Ctrl + V" e vou renomeá-la como DevSecurityConfigurations. Ela vai representar as configurações de segurança do ambiente de Dev.

[03:07] Eu criei a classe, o Spring está com aquele DevTools, ele já detectou que teve outra classe, ele reiniciou e gerou um problema.

[03:17] O problema é que eu tenho duas classes de segurança que estão configurando a mesma coisa, que é a parte de segurança. Então deu um conflito, o Spring não sabe qual das duas classes ele tem que carregar primeiro.

[03:27] Mas o ponto é que eu quero que o Spring carregue a segunda classe, a DevSecurityConfigurations apenas no ambiente de desenvolvimento.

[03:37] Para fazer isso, em cima da classe DevSecurityConfigurations, junto com as anotações que já tem, eu vou colocar mais uma anotação, que vai dizer para o Spring carregar essa classe só se o *profile* ativo no momento for esse nome que você vai passar dentro das aspas.

[03:54] Vou chamar o *profile* de Dev, então `@Profile("dev")`. Dei um "Ctrl + Shift + O" para importar a anotação *profile*. Ela veio do pacote `org.springframework.context.annotation.Profile`, então cuidado na hora que você for importar a classe. E vou salvar.

[04:09] Então com isso o Spring já sabe que só vai carregar essa classe se o *profile* ativo for o de desenvolvimento. Então ele vai reiniciar novamente.

[04:18] Só que tem um problema: eu não disse qual é o *profile* no qual a classe SecurityConfigurations deve ser carregada. Então o Spring considera que ele sempre vai carregar essa classe, e aí está o problema.

[04:32] Então na classe SecurityConfigurations eu também vou colocar o `@profile`, só que vou colocar como sendo `prod`, que seria o ambiente de produção. Então fica `@Profile("prod")`.

[04:40] Então além de configurar a DevSecurityConfigurations com o *profile* de Dev, eu tenho que dizer que a SecurityConfigurations também só deve ser carregada em determinado *profile*, que no caso é o de produção.

[04:52] Agora o Spring já sabe em qual cenário ele carrega cada uma das classes. Mas mesmo assim podemos ter um problema, porque no nosso proj

implementamos o mecanismo de autenticação e criamos a classe AutenticacaoController.

[05:08] Nessa classe estamos injetando o AuthenticationManager. Só que essa classe também é de segurança, então não quero carrega-la no ambiente de desenvolvimento.

[05:18] Então eu vou colocar `@Profile("prod")` , dizendo para o Spring só carregar esse *controller* no ambiente de produção. Se eu não estiver num ambiente de produção ele deve ignorar esse *controller* porque ele é só para um ambiente de produção.

[05:31] O Spring vai reiniciar e a ideia é que agora funcione, não é para dar nenhum erro. Ele não vai dar nenhum erro, só que nos esquecemos de fazer um detalhe. Nós dissemos para o Spring em qual cenário ele carrega cada uma daquelas classes, mas não dissemos para o Spring qual é o *profile* ativo do momento.

[05:50] Como o Spring vai saber qual é o *profile* que está ativo no momento e determinar quais classes ele vai carregar? Ele não sabe.

[05:57] E se olharmos o Console, quando o Spring inicializa o projeto, bem no começo, a segunda coisa que ele imprime é o seguinte: `The following profiles are active: dev` . Ele detectou que o *profile* atual que está sendo executado é o de desenvolvimento.

[06:20] Mas como ele sabe que eu estou num ambiente de desenvolvimento? Porque eu tinha feito no meu projeto uma configuração que eu vou mostrar para vocês. Têm várias maneiras de você dizer para o Spring qual é o *profile* ativo no momento. Nessa aula veremos uma delas e futuramente veremos as outras maneiras.

[06:38] Uma delas é você clicar com o botão direito no projeto, vir na opção "Run As > Run Configurations", e no menu à esquerda que apareceu você

confere se está selecionado “Java Application” e “Forum Application”.

[06:54] E na parte de cima, na aba “Arguments”, no campo “VM arguments” você vai colocar essa informação: `-Dspring.profiles.active=` e em seguida o *profile* que você quer que esteja ativo no momento.

[07:10] Então essa é uma variável, um parâmetro que você passa para JVM e o Spring lê essa variável. `-D` é para dizer que é um parâmetro. E cuidado que não tem espaço, é tudo junto. Em seguida a variável que o Spring vai carregar, que é a `spring.profiles.active` .

[07:27] Ele vai detectar que você passou esse parâmetro como VM argument, ele vai ler qual é o valor que você passou depois do símbolo de igual. E eu passei no caso `dev` . Então ele vai considerar que o *profile* ativo é o *profile* de dev.

[07:41] Com isso o Spring sabe que quando ele for rodar o projeto, o *profile* ativo do momento é o *profile* de dev.

[07:47] Eu vou fazer o seguinte para vocês verem: eu vou parar o projeto, vou voltar naquelas configurações clicando em "forum (forum aula_2)" com o botão direito do mouse para selecionar a opção "Run As > Run Configurations...". Na janela "Create, manage and rum configuration" vou na aba “VM arguments” e vou apagar o conteúdo do campo "VM arguments". Vou dar um “Apply” e vou clicar no “Run”. Ou seja, eu vou rodar o projeto sem configurar qual é o *profile* ativo do momento.

[08:05] E nesse caso vamos ver o que o Spring imprime no Console. Ele deveria imprimir que ele não achou o *profile* ativo. E quando você não define um *profile*, ele imprime que não achou nenhum *profile* ativo e vai usar o *profile* chamado *default*.

[08:25] Então o Springer tem um *profile* chamado *default*, que carrega todas as classes que não estão configuradas com o `@Profile` . Então se você não

configurar o *profile*, o *profile* ativo do momento será o *default*.

[08:40] Então vou voltar no Run Canfigurations e no "VM arguments" vou colar de novo aquele parâmetro `-Dspring.profiles.active.dev`, vou aplicar e rodar. Agora a mensagem que vai aparecer é a de que o *profile* ativo no momento é o de dev. Deu certo: "The following profiles are active: dev".

[09:09] Então ele já carregou aquela propriedade, carregou o perfil de dev.

[09:15] Nesse momento ele vai carregar as classes que estiverem com `@Profile("dev")` e as que não tiverem um `@Profile`, porque se não tiver ele considera que é para sempre carregar.

[09:25] A classe SecurityConfigurations, que está com `@Profile("prod")` está com um *profile*, só que é o de prod, então ele não deveria ter carregado essa classe.

[09:34] Para testar, eu dupliquei a classe DevSecurityConfigurations, mas o código dela está exatamente igual ao da SecurityConfigurations. E no ambiente de desenvolvimento eu tinha dito que a regra é que eu não quero nenhuma configuração de segurança.

[09:50] Então vou apagar todo o código da DevSecurityConfigurations, só vou deixar o método `configure`, das URL's. Todos os outros eu vou apagar. Só que eu vou apagar as URL e eu só vou deixar uma única URL.

[10:06] Não vou restringir o método, então todos os métodos vão estar valendo. Só que a URL é `("/**").permitAll()`.

[10:15] Vou tirar o `.anyRequest().authenticated()`. O do CSRF tem que deixar desligado, senão ele vai sempre procurar o Token de cross-site request forgery.

[10:25] Vou apagar as coisas da autenticação *stateless*, vou organizar os *imports* com "Ctrl + Shift + O", e pronto.

[10:32] No ambiente de desenvolvimento essa é a regra: `permitAll` para todas as URL's. Eu quero liberar todas as URL's, eu não quero ter segurança.

[10:42] Então, só para garantir, vou parar o projeto, vou rodar de novo e agora ele vai carregar o *profile* de Dev. E a configuração de segurança é que não tem nenhuma regra de autenticação e autorização, é para permitir todas as URL's.

[10:55] Então se der certo, quando dispararmos qualquer requisição para qualquer endereço, mesmo os que eram restritos, no ambiente de desenvolvimento não era para o Spring validar.

[11:06] Podemos fazer o teste no Postman, com aquele mesmo *endpoint*, DELETE para `/topicos/1`. Vou disparar a requisição. Quando ele terminar de carregar e aquela requisição for disparada é para o Spring dar um HTTP 200, não é para dar mais aquele 403 "forbidden", porque agora não é mais para exigir autenticação e muito menos autorização.

[11:33] Vamos testar novamente, agora ele já carregou. Está disparando a requisição no Postman, e espero que apareça o código 200 e tenha dado certo o nosso código, que ele tenha só ignorado todas as partes de autenticações. E funcionou.

[11:54] Então vamos listar, só para garantir mesmo. Vou disparar um GET para `/topicos`. Só é para vir o tópico de id 2 e o de id 3, porque o de id 1 eu acabei de excluir e não reiniciei o projeto. Funcionou.

[12:08] Não estou autenticado, na aba "Headers" não tem nada, na aba "Authorization" não tem nada. Então realmente, no ambiente de desenvolvimento agora eu não preciso mais estar logado, fica muito mais simples, muito mais produtivo. Mas no ambiente de produção vai ter as configurações de segurança.

[12:26] Então esse era o objetivo da aula de hoje, mostrar um exemplo de utilização dessa questão do *profile* para configurar um tipo de classe que é

carregada baseada num tipo de *profile*.

[12:35] Mais para frente veremos a parte de Deploy e outras questões de configurações do "application.properties", como ele funciona no caso de *profiles*. Mas isso futuramente na aula de Deploy veremos com calma.

[12:48] Eu espero que vocês tenham gostado da aula de hoje. Vejo vocês no próximo vídeo. Um abraço e até lá.