09

Validação e criptografia

Transcrição

[00:00] No vídeo anterior fizemos a lógica de cadastrão de usuário, mas de um modo simples, sem validação e segurança. Vamos resolver esses problemas agora, primeiro vamos fazer aquela validação básica com as anotações do play, todos os campos vão ser obrigatórios, então @Required do pacote constraints do play, vamos passar a mensagem que é “você precisa fornecer um nome de tratamento”.

[00:33] Para o email e para senha a mesma coisa, só que muda aqui que você precisa fornecer um email e que você precisa fornecer uma senha, agora vamos lá no UsuarioController e fazemos a validação, caso existam erros no formulário, formulario.hasErros, rejeitamos o formulário e retornamos ele de volta para tela.

[01:07] Vamos primeiro mandar uma mensagem para o usuário, flash, danger que é o tipo de erro do bootstrap, “Hà erros no formulário de cadastro”. E agora retornamos o formulário já preenchido, return badRequest, com o nosso formulário renderizado, então formularioDeNovoUsuario.render com o formulário que já temos preenchido aqui na nossa tela.

[01:37] Agora que o nosso formulário já tem algumas validações tanto Java quanto Html, lembrando que o play já adiciona diretamente essa restrição no campo usando os ajudantes, temos que lembrar que toda vez que você cadastra um usuário e pede uma senha para ele, você também pede uma confirmação para garantir que ele sabe qual é a senha que ele está digitando. Então vamos lá, vamos criar esse campo.

[02:03] Vamos vir aqui, eu vou copiar o campo da senha e eu vou mudar aqui para “Confirmação de senha”, essa é a label, aqui vamos ter confirmaSenha. Mas esse campo não existe no modelo, não existe, mas também não precisa, podemos pegar esse dado do request mesmo sem ele estar no nosso modelo. Vamos ver como é isso, vamos criar o validador de senha, para isso vamos criar um validador de usuário realmente?

[02:35] Vamos lá, eu vou injetar o validador de usuário “private ValidadorDeUsuario”, essa classe ainda não existe então vamos criar ela, create class, no package validadores, salva ela, volta aqui para o controller, salva novo usuário e em vez de perguntar pro formulário de erro, vamos perguntar para o validador se existem erros no formulário, ValidadorDeUsuario.temErros(formulário).

[03:10] Fizemos igualzinho como tínhamos feito no produto, vamos criar esse método e vamos validar a nossa senha, então vamos lá, primeiro temos que pegar a senha, então é formulario.field(“senha”), e pegamos o valor dela, só que eu não quero ficar comparando se a senha é nula ou não, então eu vou aqui passar um valor padrão, caso seja nula ele retorna uma senha vazia, eu vou guardar isso em uma variável chamada senha, a mesma coisa com a confirmação da senha.

[03:55] Podemos vir aqui e pegar o valor do confirmaSenha, mesmo que ela não exista no modelo, temos a senha confirma senha, só que não quero ficar jogando um monte de coisa aqui, então vou extrair isso para um método, o método se chama validaSenha e vai receber um formulário, e aqui o método temErros vai retornar se o formulario.hasErros, então se o nosso formulário tiver erros, que vamos criar erros aqui rejeitando eles, ele vai retornar um true falando que o formulário tem erros.

[04:32] Qual a primeira validação? Já fizemos a validação de que a senha não está vazia, vamos fazer isso para confirmação de senha também, if (confirmaSenha. isEmpty()), ela nunca vai ser nula porque ela sempre vai retornar algum valor, rejeitamos o campo da confirmação de senha, então “formulario.reject(new ValidationError(“confirmaSenha”, “Você precisa fornecer uma confirmação de senha”)”.

[05:12] Se a confirmação de senha existir, a senha também já existe, então fazemos um else if a senha for igual a confirmação de senha, !senha.equals((confirmaSenha)), só que aqui se ela não for igual para podermos rejeitar. Se a senha não for igual para confirmação, rejeitamos o campo também, "formulario.reject(new ValidationError("senha", "")", já explico depois o porquê, e eu vou rejeitar o confirmaSenha também e vou falar "as senhas precisam ser iguais".

[06:05] Validamos a nossa senha, vou dar um save aqui e vamos testar aqui. Vamos lá no usuario/novo, vou tentar criar um usuário chamado marco, com email marco@caelum.com.br, com a senha marco e a contra-senha marco1, sem querer digitei o 1. Olha só, ele colore o campo senha, colore o campo confirmação e mostra a mensagem aqui em baixo, se eu tivesse colocado a mensagem na senha, ele mostraria uma mensagem aqui no meio e ficaria meio feio, ou ficaria duplicado o que não é legal, assim fica bonitinho, então está bom.

[07:00] O que falta confirmar? Falta confirmar o email, então primeiro deixa eu confirmar aqui que isso está funcionando, confirmei, consegui cadastrar meu usuário, dei um select com a senha aqui, existe um usuário com a senha marco explicita assim no banco não é legal. Vamos criptografar ela, vamos lá antes de salvar o usuário vamos pegar a senha dele e criptografar, como? Com a classe Crypt do pacote akka.util, vamos usar o algoritmo sha1 e passar a senha do usuário, usuario.getSenha().

[07:41] Guarda isso em uma variável, senhaCrypto e jogamos ela de volta no nossos usuário, usuario.setSenha(senhaCrypto), então de salvar criptografamos a senha e devolvemos ela no modelo. Vamos tentar de novo, eu vou voltar aqui, usuario/novo vou salvar o mesmo usuário, marco com marco@caelum.com.br, com a senha marco e confirmação de senha marco, parece que deu tudo certo, vamos ver aqui no terminal, vamos dar outro select, senha criptografada bonitinho.

[08:25] Só que eu tenho dois usuários duplicados com mesmo email, vamos garantir que não exista essa duplicação de e-mails, então próximo passo validar email, vamos no validador de usuário vamos criar um novo método chamado validaEmail, que recebe também o formulário para extrairmos os dados dele lá dentro.

[08:50] Para validar o email, precisamos fazer consulta no banco, então vamos precisar de um usuário DAO, ainda não temos ele, então vou injetar ele aqui, "private UsuarioDAO usuarioDAO". Vamos criar essa classe, eu já tinha criado essa classe, então basta para gente criar os métodos dela, você pode ver aqui que ela está vazia.

[09:18] Então vamos lá, validaEmail, primeiro pegamos o método do email, formulario.field ("email").valueOr(" "), guarda isso em uma variável chamada email, e vamos confirmar aqui que não existe um usuário com esse email, então if(usuarioDAO.comEMail(email).IsPresent()), o que isso vai fazer? Vamos retornar um optional que é o nosso padrão usando o Java 8, então vamos criar esse método aqui com email, optional de usuário e temos que fazer uma lógica aqui para ele retornar esse objeto.

[10:07] Primeiro precisamos buscar no banco, para isso precisamos de um finder, então private finder<Long, Usuario> usuarios = new Finder<>(Usuario.class). Já temos nosso buscador, é só buscar no banco, usuarios.where, vamos pegar onde o email seja igual ao nosso email que recebemos aqui no método, e temos que achar um único usuário que tenha esse email, vou guardar ele em uma variável usuário, e retornamos aqui um objeto Opitional.ofNullabeble(usuário) porque esse objeto pode vir nulo.

[11:10] Agora só falta, caso exista um usuário com esse email, rejeitamos, então formulario.reject(new ValidationError("email", "Já existe um usuário cadastrado com esse email")). Terminamos a nossa validação de senha e de email, acho que é isso, vamos fazer o teste.

[11:45] Vamos vir aqui, vamos tentar criar um novo usuário, vamos em usuario/novo, primeiro eu vou apagar aqueles usuários ali do banco e eu vou criar dois, então agora não temos nenhum. Vamos lá, vou criar dois usuário com mesmo email, marco, email marco@caelum.com.br, senha marco, confirmação ok, agora vou criar um novo marco@caelum.com.br, eu vou até colocar marco2 aqui, senha marco2 para garantir que não é esse o problema.

[12:20] Agora há erros no formulários já existe um usuário cadastrado com esse email, sucesso. Aprendemos nesse vídeo a utilizar formulários de modelos, um formulário de usuário, só que com campos dinâmicos, que nem aquele confirmação de senha, que não existe no modelo mas ainda conseguimos pegar o dado da request e aprendemos também a usar a criptografia da classe crypto do pacote do akka.

[12:46] Só que tem um problema aqui, eu posso criar um usuário por exemplo, do Guilherme, e eu vou ter acesso ao email do Guilherme, porque eu vou ser o único que sabe a senha dele, isso não é legal, o que os sistemas fazem em geral é enviar um email para o usuário clicar em um link que confirma que aquele email pertence a ele certo? Vamos fazer isso na próxima aula.