10

Validação e criptografia

Temos um cadastro de usuários funcional, mas ainda não fazemos validação alguma. Primeiramente, vamos fazer a validação básica com as anotações do Play!. Adiciona na classe `Usuario` as seguintes restrições.

```
@Required(message = "Você precisa fornecer um nome de tratamento")
private String nome;
@Email @Required(message = "Você precisa fornecer seu email")
private String email;
@Required(message = "Você precisa fornecer uma senha")
private String senha;
```

Agora, antes de salvar o usuário, conferimos se há erros no formulário e, caso existam, mostramos uma mensagem de erro e renderizamos o formulário novamente, já preenchido.

```
public Result salvaNovoUsuario() {
    //...
    if (formulario.hasErrors()) {
        flash("danger", "Há erros no formulário de cadastro de novo usuário");
        return badRequest(formularioDeNovoUsuario.render(formulario));
    }
    //...
}
```

Agora que o formulário já tem algumas validações tanto em Java quanto em HTML, precisamos criar um campo de confirmação de senha.

```
@b3.password(formulario("confirmaSenha"), '_label -> "Confirmação de senha", 'required -> true)
```

Os campos de nossos formulários não precisam ser todos restritos aos atributos to modelo. Podemos criar novos campos e acessá-los depois no controller. A sintaxe, no entanto, será um pouco diferente.

Agora precisamos fazer as validações de usuário relativas a email e senha. Criamos então um validador de usuário e usamos ele ao invés do formulário.

```
@Inject private ValidadorDeUsuario validadorDeUsuario;
public Result salvaNovoUsuario() {
    //...
    if (validadorDeUsuario.temErros(formulario)) { ... }
    //...
}
```

Agora quanto à classe do validador em si, vamos separar a cada tipo de validação em um método diferente.

```
package validadores;
import play.data.*;
```

```
import models.Usuario;
public class ValidadorDeUsuario {
    @Inject private UsuarioDAO usuarioDAO;
    public boolean temErros(Form<Usuario> formulario) {
        validaEmail(formulario);
        validaSenha(formulario);
        return formulario.hasErrors();
    }
    private void validaEmail(Form<Usuario> formulario) {
        Usuario usuario = formulario.get();
        if (usuarioDAO.comEmail(usuario.getEmail()).isPresent()) {
            formulario.reject(new ValidationError("email", "Este email já foi cadastrado"));
        }
    }
    private void validaSenha(Form<Usuario> formulario) {
        String senha = formulario.field("senha").valueOr("");
        String confirmacao = formulario.field("confirmaSenha").valueOr("");
        if (confirmacao.isEmpty()) {
            formulario.reject(new ValidationError("confirmaSenha", "Você precisa fornecer uma c
        } else if (!senha.equals(confirmacao)) {
            formulario.reject(new ValidationError("senha", ""));
            formulario.reject(new ValidationError("confirmaSenha", "As senhas precisam ser igua
        }
    }
}
```

Como precisamos acessar o banco para conferir se já existe um usuário cadastrado com o email fornecido, precisamos criar um novo DAO.

```
package daos;
import ...;
public class UsuarioDAO {
    private Finder<Long, Usuario> usuarios = new Finder<>(Usuario.class);
    public Optional<Usuario> comEmail(String email) {
        Usuario usuario = usuarios.where().eq("email", email).findUnique();
        return Optional.ofNullable(usuario);
    }
}
```

E enfim, com um usuário validado, podemos encriptar a senha logo antes de salvar no banco.

```
public Result salvaNovoUsuario() {
    //...
    Usuario usuario = formulario.get();
    String criptoSenha = akka.util.Crypt.sha1(usuario.getSenha());
    usuario.setSenha(criptoSenha);
    usuario.save();
    //...
}
```