02

Preparando modelo e view

Transcrição

Vamos continuar melhorando a nossa aplicação. Nessa aula iremos falar sobre autenticação e autorização.

Por exemplo, quando tentamos acessar página da Alura (https://cursos.alura.com.br/dashboard (https://cursos.alura.com.br/dashboard)), somos apresentados a uma página de login.

Queremos fazer algo parecido no nosso sistema, que até o momento pode ser acessado livremente desde que o servidor esteja rodando.

Para isso teremos que aumentar a complexidade do nosso **Modelo**, pois, além da empresa, teremos que representar um usuário no sistema. Como é uma nova funcionalidade, também será necessário trabalharmos com o **Controlador** e o **View**.

De início, criaremos uma classe para representar o usuário no nosso pacote `modelo`. Para não perdermos tempo construindo esse código simples, já preparamos essa classe previamente:

```
public class Usuario {

    private String login;
    private String senha;

    public String getLogin() {
```

```
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public boolean ehIgual(String login, String senha) {
        if(!this.login.equals(login)) {
            return false;
        }

        if(!this.senha.equals(senha)) {
            return false;
        }

        return true;
    }
}
```

COPIAR CÓDIGO

Portanto, basta copiar esse código e colar na classe `Usuario` que foi criada. Nesse caso definimos apenas **login** e **senha**, mas em um sistema real teríamos mais informações, como e-mail, endereço, plano, etc.

Cada um desses atributos tem um `get()` e um `set()`. Além deles, criamos outro método, chamado `ehIgual()` (parecido com o método `equals()`), que recebe `login` e `senha`. A partir desse método poderemos verificar se o objet

usuário tem as mesmas credenciais (login e senha). Se forem iguais, o método devolve `true`, do contrário ele devolve `false`.

Agora precisamos preparar a "persistência" no nosso `Banco`, pois ele precisa saber quais usuários existem. Antes de fazer login na Alura, o usuário precisa fazer um cadastro. Ou seja, além das empresas no nosso banco, precisamos de uma nova lsita para usuários:

```
private static List<Usuario> listaUsuarios= new ArrayList<>();
```

COPIAR CÓDIGO

No bloco seguinte, que é automaticamente executado, vamos simular a criação de dois usuários, `nico` e `ana`. Depois, adicionaremos esses usuários na lista:

```
static {
    Empresa empresa = new Empresa();
    empresa.setId(chaveSequencial++);
    empresa.setNome("Alura");
    Empresa empresa2 = new Empresa();
    empresa2.setId(chaveSequencial++);
    empresa2.setNome("Caelum");
    lista.add(empresa);
    lista.add(empresa2);

    Usuario u1 = new Usuario();
    u1.setLogin("nico");
    u1.setSenha("12345");

    Usuario u2 = new Usuario();
    u2.setLogin("ana");
    u2.setSenha("12345");

    listaUsuarios.add(u1);
    listaUsuarios.add(u2);
```

```
}
```

COPIAR CÓDIGO

Assim, nossos usuários são criados da mesma forma que criamos as empresas, e então são adicionados ao nosso banco. Mais tarde criaremos um método que busca o usuário.

Antes disso, se queremos criar uma autenticação de usuário, precisaremos criar um formulário de login - nesse caso, um `.jsp`. Para isso, trabalharemos em cima do formulário que criamos anteriormente para `NovaEmpresa` - afinal, um formulário é um formulário, quer seja para a criação de uma empresa ou para fazer login.

Portanto, vamos copiar e colar `formNovaEmpresa.jsp`, e chamaremos esse novo `.jsp` de `formLogin`. Esse formulário terá um `login` e uma `senha`, e esses serão os nomes dos parâmetros.

Existem vários tipos de `input` em Java. Como queremos que o parâmetro `senha` fique escondido pelo navegador, definiremos seu `type` como `password`.

Além disso, precisamos definir o nome da ação - no caso, `Login`.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %
<c:url value="/entrada" var="linkEntradaServlet"/>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

```
        <form action="${linkEntradaServlet }" method="post">

            Login: <input type="text" name="login"  />
            Senha: <input type="password" name="senha"  />

            <input type="hidden" name="acao" value="Login" />

            <input type="submit" />
        </form>

    </body>
    </html>
```

COPIAR CÓDIGO

Agora precisamos criar a ação `Login`. Para isso, criaremos uma nova classe no pacote `acao` clicando com o botão direito e em seguida em "New > Class".

Na lateral direita da janela de diálogo que é exibida, temos a opção de implementar uma interface clicando em `Add`. Na próxima janela, basta selecionarmos s a interface `Acao`, clicarmos em "OK" e em seguida em "Finish". Dessa forma, o Eclipse irá preencher automaticamente o código com `implements Acao` e o método que precisamos implementar:

```
public class Login implements Acao {

    @Override
    public String executa(HttpServletRequest request, HttpServl
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        return null;
    }

}
```

COPIAR CÓDIGO

Quando o usuário enviar os dados na nossa página, ele irá submeter uma requisição que cai na ação `Login`. Portanto, primeiramente, precisamos ler os parâmetros `login` e `senha`. Faremos isso com `String login = request.getParameter("login")`. Repetiremos o mesmo processo para `senha`.

Para acompanharmos o que acontece no nosso código, criaremos um `System.out.println("Logando " + login)`.

Vamos pensar em um cenário no qual o login funcionou. Nesse caso, o usuário deverá ser redirecionado para algum lugar. Como não temos uma dashboard, vamos redirecioná-lo para `ListaEmpresas`.

Com isso, teremos:

```
public class Login implements Acao {

    @Override
    public String executa(HttpServletRequest request, HttpServ
            throws ServletException, IOException {

        String login = request.getParameter("login");
        String senha = request.getParameter("senha");

        System.out.println("Logando " + login);


        return "redirect:entrada?acao=ListaEmpresas";
    }

}
```

COPIAR CÓDIGO

Pode ser que ainda esteja faltando algo, mas nada nos impede de testarmos o nosso código, certo? Para isso, precisaremos chamar o `.jsp` para exibirmos

formulário de Login. Nosso `Login.jsp` está na pasta `WEB-INF`, mas está escondido. Portanto, não conseguiremos chamar o `jsp` diretamente, e não temos uma ação para fazer essa tarefa.

Como nossa ação `Login` é executada a partir do formulário, primeiro precisaremos chegar no formulário a partir de outra ação, sempre seguindo o modelo do nosso projeto - ou seja, passando pelo controlador.

Em vista disso, criaremos uma nova classe `LoginForm` no nosso pacote `acao`, já selecionando nossa interface `Acao`. Como essa ação só é responsável por chamar o formulário, escreveremos apenas `return forward:formLogin.jsp`:

```
public class LoginForm implements Acao {

    @Override
    public String executa(HttpServletRequest request, HttpServl
            throws ServletException, IOException {

        return "forward:formLogin.jsp";
    }

}
```

COPIAR CÓDIGO

Repare que fizemos tudo isso sem alterar qualquer coisa no controlador. Agora vamos iniciar o Tomcat e acessar a URL http://localhost:8080/gerenciador/entrada?acao=LoginForm (http://localhost:8080/gerenciador/entrada?acao=LoginForm). Veremos que nosso formulário foi carregado corretamente.

Vamos observar o código fonte para verificarmos o que o navegador recebeu:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <form action="/gerenciador/entrada" method="post">

        Login: <input type="text" name="Login"  />
        Senha: <input type="passwaord" name="senha"  />

        <input type="hidden" name="acao" value="Login" />

        <input type="submit" />
    </form>

</body>
</html>
```

COPIAR CÓDIGO

Aparentemente tudo está correto. Além disso, quando fazemos o login com o usuário `nico` e a senha `12345` (que, como pretendíamos, não é exibida pelo navegador), somos redirecionados para `ListaEmpresas`.

Já conseguimos implementar um formulário de login. No entanto, se reiniciarmos o servidor e tentarmos acessar a URL http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas (http://localhost:8080/gerenciador/entrada?acao=ListaEmpresas), continuaremos tendo acesso a ela, sem precisarmos fazer login novamente.

Ou seja, apesar de termos um login, ainda não fechamos a nossa aplicação e não verificaremos se o usuário fez login ou não. Começaremos essa lógica isso no próximo vídeo. Até lá!