02

Cadastro de usuários

Agora que a página inicial da aplicação está desenvolvida, vamos trabalhar no layout que será utilizado pela aplicação. Para conseguirmos um layout reutilizável, criaremos duas novas jsps que serão utilizadas no código das views, `header.jsp` e `footer.jsp`. Esses arquivos devem ser criados dentro da pasta `WEB-INF/jsp` do projeto.

```
// header.jsp
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
    <title>Título</title>
    <link href="<c:url value='/css/bootstrap.css'/>" rel="stylesheet" />
    <link href="<c:url value='/css/site.css'/>" rel="stylesheet" />
</head>
<body>
    <header>

    </header>
    <nav>
        <ul class="nav nav-tabs">
            <li><a href="<c:url value='/' />">Home</a></li>
            <li><a href="<c:url value='/usuario/lista' />">Usuarios</a></li>
            <li><a href="<c:url value='/horaLancada/lista' />">Horas Cadastradas</a></li>
            <li><a href="<c:url value='/horaLancada/form' />">Cadastrar Horas</a></li>
        </ul>
    </nav>
    <div class="container">
        <main class="col-sm-8">


// footer.jsp

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
        </main>
    </div>
</body>
</html>
```

Podemos importar o cabeçalho e o rodapé utilizando a tag `import` da JSTL no código da view:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:import url="/WEB-INF/jsp/header.jsp"/>

Bem-vindos à aplicação de cadastro de horas

<c:import url="/WEB-INF/jsp/footer.jsp"/>
```

Cadastro de usuários

O próximo passo na aplicação é desenvolver a tela de cadastro de usuários no banco de dados. Vamos começar implementando a classe que acessará as informações da tabela de usuários, o `UsuarioDAO` :

```
public class UsuarioDAO{
   public void adiciona(Usuario usuario){

   }

   public List<Usuario> lista(){

   }
}
```

Para implementarmos os métodos desse DAO, precisaremos do Entity Manager da JPA. Como não queremos fazer o gerenciamento manual do tempo de vida da conexão, configuraremos um producer do CDI para termos injeção de dependências no DAO.

```
@ApplicationScoped
public class EntityManagerProducer{
    private static EntityManagerFactory factory =
        Persistence.createEntityManagerFactory("default");

    @Produces @RequestScoped
    public EntityManager getEntityManager(){
        return factory.createEntityManager();
    }

    public void close(@Disposes EntityManager manager){
        manager.close();
    }
}
```

Com o producer criado, podemos utilizar o CDI para fazer injeção de dependências no construtor do DAO:

```
@RequestScoped
public class UsuarioDAO{
    private EntityManager manager;

    public UsuarioDAO(){}

    @Inject
    public UsuarioDAO(EntityManager manager){
        this.manager = manager;
    }

    public void adiciona(Usuario usuario){
        manager.getTransaction().begin();
        manager.persist(usuario);
        manager.getTransaction().commit();
    }

    public List<Usuario> lista(){
        String jpql = "select u from Usuario u";
```

```
        TypedQuery<Usuario> query = manager.createQuery(jpql, Usuario.class);
        return query.getResultList();
    }
}
```

O próximo passo é criar um controller que gerenciará os usuários da aplicação:

```
@Controller
public class UsuarioController{
    private Result result;

    private UsuarioDAO usuarioDAO;

    private Validator validator;

    public UsuarioController(){}

    @Inject
    public UsuarioController(Result result, UsuarioDAO usuarioDAO, Validator validator){
        this.result = result;
        this.usuarioDAO = usuarioDAO;
        this.validator = validator;
    }

    public void form(){}

    @IncludeParameters
    @Post
    public void adiciona(@Valid Usuario usuario){
        validator.onErrorRedirectTo(this).form();
        usuarioDAO.adiciona(usuario);
        result.redirectTo(this).lista();
    }

    public void lista(){
        result.include("usuarios", usuarioDAO.lista());
    }
}
```

Para terminarmos, precisamos apenas criar os arquivos de visualização para as lógicas que foram colocadas no controller. Começaremos pelo código do formulário de cadastro (`WEB-INF/jsp/usuario/form.jsp`):

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:import url="/WEB-INF/jsp/header.jsp"/>
<form action="${linkTo[UsuarioController].adiciona(null)}" method="post">
    <label for="nome">Nome:</label>
    <input type="text" id="nome" name="usuario.nome" class="form-control" />

    <label for="email">E-mail:</label>
    <input type="text" id="email" name="usuario.email" class="form-control" />

    <label for="login">Login:</label>
    <input type="text" id="login" name="usuario.login" class="form-control" />
```

```
        <label for="senha">Senha:</label>
        <input type="password" id="senha" name="usuario.senha" class="form-control" />

        <input type="submit" value="Cadastrar" class="btn"/>
    </form>
    <c:import url="/WEB-INF/jsp/footer.jsp"/>
```

Se o usuário preencher dados inválidos nesse formulário de cadastro, queremos mostrar o formulário preenchido com as informações que foram enviadas para o servidor e as mensagens de validação. Para podermos aplicar estilo nos erros de validação, colocaremos as mensagens dentro de `span` s com a classe `validation-error` . Então para mostrarmos a mensagem de validação do campo `usuario.nome` , utilizaríamos o seguinte código:

```
<span class="validation-error">
   ${errors.from("usuario.nome")}
</span>
```

Precisaremos desse código para cada um dos campos do formulário. Para evitar a replicação, podemos isolar esse código que gera as mensagens de validação dentro do `span` dentro de uma nova tag customizada do projeto. Dentro da pasta `WEB-INF/tags` criaremos um novo arquivo chamado `validationMessage.tag` com o seguinte código:

```
<%@ attribute name="name" required="true" %>
<span class="validation-error">${errors.from(name)}</span>
```

O código final do formulário com as mensagens de validação fica da seguinte forma:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib tagdir="/WEB-INF/tags" prefix="alura" %>
<c:import url="/WEB-INF/jsp/header.jsp"/>
<form action="${linkTo[UsuarioController].adiciona(null)}" method="post">
    <label for="nome">Nome:</label>
    <input type="text" id="nome" name="usuario.nome" class="form-control" value="${usuario.nome
    <alura:validationMessage name="usuario.nome"/>

    <label for="email">E-mail:</label>
    <input type="text" id="email" name="usuario.email" class="form-control" value="${usuario.ema
    <alura:validationMessage name="usuario.email"/>

    <label for="login">Login:</label>
    <input type="text" id="login" name="usuario.login" class="form-control" value="${usuario.log
    <alura:validationMessage name="usuario.login"/>

    <label for="senha">Senha:</label>
    <input type="password" id="senha" name="usuario.senha" class="form-control" />
    <alura:validationMessage name="usuario.senha"/>

    <input type="submit" value="Cadastrar" class="btn"/>
</form>
<c:import url="/WEB-INF/jsp/footer.jsp"/>
```

Para terminarmos, precisamos criar a view para a lista de usuários cadastrados. Dentro da pasta `WEB-INF/jsp/usuario` crie o arquivo `lista.jsp` :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:import url="/WEB-INF/jsp/header.jsp"/>

<a href="${linkTo[UsuarioController].form()}">Novo usuário</a>
<table class="table table-hover">
    <thead>
        <tr>
            <th>Id</th>
            <th>Nome</th>
            <th>E-mail</th>
            <th>Login</th>
        </tr>
    </thead>
    <tbody>
        <c:forEach items="${usuarios}" var="u">
            <tr>
                <td>${u.id}</td>
                <td>${u.nome}</td>
                <td>${u.email}</td>
                <td>${u.login}</td>
            </tr>
        </c:forEach>
    </tbody>
</table>

<c:import url="/WEB-INF/jsp/footer.jsp"/>
```