Java JSP – CRUD utilizando BD h2

Neste artigo, vamos criar um projeto CRUD (Create, Read, Update, Delete) completo em Java com JSP (JavaServer Pages), H2 (banco de dados em memória) e NetBeans como IDE.

Neste projeto incluirá funcionalidades de login e, após autenticação, o usuário será redirecionado para a tela de CRUD onde poderá realizar as operações de **criar**, **listar**, **editar** e **excluir** usuários.

Passo 1: Criar o Projeto no NetBeans

1. **Abra o NetBeans** e selecione **File > New Project**.
2. Selecione **Java Web > Web Application** e clique em **Next**.
3. Dê o nome ao projeto, por exemplo, CrudJSPExample, e clique em **Finish**.
4. Escolha o servidor **Apache Tomcat** (ou outro de sua preferência) e clique em **Finish**.

Passo 2: Configurar o Banco de Dados H2

1. **Criação do arquivo de configuração do H2**:

   O banco de dados H2 será configurado no arquivo persistence.xml, localizado dentro da pasta WEB-INF/classes/META-INF.

2. **Conteúdo do arquivo persistence.xml**: Esse arquivo configura a conexão com o banco de dados, utilizando o H2 como banco em memória e o Hibernate para gerenciamento de persistência.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                                 http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
  <persistence-unit name="crudPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>model.User</class> <!-- Classe do modelo de dados -->
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
```

```
            <property name="hibernate.hbm2ddl.auto" value="update"/> <!-- Criação
        automática das tabelas -->
            <property name="hibernate.show_sql" value="true"/> <!-- Exibição das SQLs
        executadas -->
            <property name="hibernate.connection.driver_class" value="org.h2.Driver"/>
            <property name="hibernate.connection.url"
        value="jdbc:h2:mem:test;DB_CLOSE_DELAY=-1"/> <!-- Banco em memória -->
            <property name="hibernate.connection.username" value="sa"/>
            <property name="hibernate.connection.password" value=""/>
        </properties>
    </persistence-unit>
</persistence>
```

Passo 3: Criar as Classes Java

3.1 Classe User.java (Modelo de Dados)

A classe User representa um usuário do sistema, contendo o nome de usuário e a senha.

Ela será anotada com JPA (@Entity) para permitir o mapeamento para o banco de dados.

```
package model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity // Define que essa classe será uma entidade do banco de dados
public class User {

    @Id // Define que o campo 'username' será a chave primária
    private String username;
    private String password;

    // Getters e Setters para acesso e manipulação dos atributos

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

3.2 Classe UserDAO.java (Acesso a Dados)

A classe UserDAO encapsula a lógica de persistência de dados, utilizando o Hibernate para salvar, editar, excluir e buscar usuários.

```
package dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import model.User;
import java.util.List;

public class UserDAO {
    // Criação do EntityManagerFactory para interação com o banco
    private static EntityManagerFactory emf =
Persistence.createEntityManagerFactory("crudPU");

    // Método para autenticar o usuário no sistema
    public static boolean authenticate(String username, String password) {
        EntityManager em = emf.createEntityManager();
        try {
            // Busca o usuário no banco usando a chave primária (username)
            User user = em.find(User.class, username);
            // Verifica se o usuário existe e a senha está correta
            return user != null && user.getPassword().equals(password);
        } finally {
            em.close(); // Fecha o EntityManager
        }
    }

    // Método para criar um novo usuário
    public static void createUser(String username, String password) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin(); // Inicia uma transação
            User user = new User();
            user.setUsername(username);
            user.setPassword(password);
            em.persist(user); // Persiste o novo usuário no banco
            em.getTransaction().commit(); // Comita a transação
        } finally {
            em.close(); // Fecha o EntityManager
        }
    }

    // Método para listar todos os usuários
    public static List<User> listUsers() {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createQuery("SELECT u FROM User u", User.class).getResultList();
        } finally {
            em.close(); // Fecha o EntityManager
        }
    }

    // Método para editar um usuário existente
    public static void updateUser(String username, String newPassword) {
        EntityManager em = emf.createEntityManager();
```

```
        try {
            em.getTransaction().begin(); // Inicia uma transação
            User user = em.find(User.class, username);
            if (user != null) {
                user.setPassword(newPassword); // Atualiza a senha do usuário
                em.getTransaction().commit(); // Comita a transação
            }
        } finally {
            em.close(); // Fecha o EntityManager
        }
    }

    // Método para excluir um usuário
    public static void deleteUser(String username) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin(); // Inicia uma transação
            User user = em.find(User.class, username);
            if (user != null) {
                em.remove(user); // Remove o usuário do banco
                em.getTransaction().commit(); // Comita a transação
            }
        } finally {
            em.close(); // Fecha o EntityManager
        }
    }
}
```

Passo 4: Criar as Páginas JSP

4.1 Tela de Login (login.jsp)

A tela de login captura o nome de usuário e senha e envia os dados ao servidor para autenticação.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <!-- Formulário para envio de dados -->
    <form action="login" method="post">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username" required><br><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password" required><br><br>
        <input type="submit" value="Login">
    </form>
    <!-- Mensagem de erro caso o login falhe -->
    <c:if test="${param.error != null}">
        <p style="color:red;">Invalid credentials. Please try again.</p>
    </c:if>
</body>
</html>
```

4.2 Tela de CRUD (crud.jsp)

A tela de CRUD permite criar, listar, editar e excluir usuários.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>CRUD Operations</title>
</head>
<body>
    <h2>CRUD Operations</h2>

    <!-- Formulário para adicionar um novo usuário -->
    <h3>Add User</h3>
    <form action="addUser" method="post">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username" required><br><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password" required><br><br>
        <input type="submit" value="Add User">
    </form>

    <h3>List of Users</h3>
    <table border="1">
        <tr>
            <th>Username</th>
            <th>Actions</th>
        </tr>
        <c:forEach var="user" items="${users}">
            <tr>
                <td>${user.username}</td>
                <td>
                    <a href="editUser?username=${user.username}">Edit</a> |
                    <a href="deleteUser?username=${user.username}">Delete</a>
                </td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Passo 5: Criar os Servlets

5.1 Servlet de Login (LoginServlet.java)

Este servlet processa a autenticação do usuário.

```
package servlet;

import dao.UserDAO;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class LoginServlet extends HttpServlet {
    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Verifica se as credenciais estão corretas
        if (UserDAO.authenticate(username, password)) {
            // Redireciona para a tela de CRUD
            response.sendRedirect("crud.jsp");
        } else {
            // Se falhar, redireciona de volta para o login com mensagem de erro
            response.sendRedirect("login.jsp?error=true");
        }
    }
}
```

5.2 Servlet para Adicionar Usuário (**AddUserServlet.java**)

Este servlet lida com a criação de um novo usuário.

```
package servlet;

import dao.UserDAO;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class AddUserServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Cria um novo usuário no banco de dados
        UserDAO.createUser(username, password);
        // Redireciona para a página de CRUD
        response.sendRedirect("crud.jsp");
    }
}
```

5.3 Servlet para Editar Usuário (**EditUserServlet.java**)

Este servlet lida com a edição de um usuário existente.

```
package servlet;

import dao.UserDAO;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class EditUserServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String newPassword = request.getParameter("password");
```

```
        // Atualiza a senha do usuário
        UserDAO.updateUser(username, newPassword);
        // Redireciona para a página de CRUD
        response.sendRedirect("crud.jsp");
    }
}
```

5.4 Servlet para Excluir Usuário (DeleteUserServlet.java)

Este servlet lida com a exclusão de um usuário.

```
package servlet;

import dao.UserDAO;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class DeleteUserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");

        // Exclui o usuário do banco de dados
        UserDAO.deleteUser(username);
        // Redireciona para a página de CRUD
        response.sendRedirect("crud.jsp");
    }
}
```

Passo 6: Configuração do web.xml

O arquivo web.xml define os servlets e suas URLs.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" version="3.0">
    <servlet>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>servlet.LoginServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>AddUserServlet</servlet-name>
        <servlet-class>servlet.AddUserServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddUserServlet</servlet-name>
        <url-pattern>/addUser</url-pattern>
    </servlet-mapping>
```

```
    <servlet>
        <servlet-name>EditUserServlet</servlet-name>
        <servlet-class>servlet.EditUserServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EditUserServlet</servlet-name>
        <url-pattern>/editUser</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>DeleteUserServlet</servlet-name>
        <servlet-class>servlet.DeleteUserServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DeleteUserServlet</servlet-name>
        <url-pattern>/deleteUser</url-pattern>
    </servlet-mapping>
</web-app>
```

Passo 7: Executar o Projeto

1. Compile e execute o projeto no NetBeans.
2. Acesse a página de login http://localhost:8080/CrudJSPExample/login.jsp.
3. Faça login com as credenciais. Após o login bem-sucedido, você será redirecionado para a tela de CRUD, onde poderá adicionar, listar, editar e excluir usuários.

Com esse passo a passo, você criou um sistema CRUD completo com login, utilizando Java com JSP, H2 e NetBeans.

O sistema permite realizar operações de **criação**, **leitura**, **atualização** e **exclusão** de usuários, com um banco de dados em memória e uma interface simples e eficaz.

Esse exemplo pode ser expandido para outras funcionalidades conforme necessário.

EducaCiência FastCode para a comunidade