Apostila: Sistema de Gestão de Usuários com Login Personalizado

Objetivo: Criar um sistema onde seja possível cadastrar novos usuários e realizar login/logout utilizando credenciais salvas no banco de dados (MySQL).

Estrutura de Pacotes:

• br.com.curso.demo.model (Entidades)

• br.com.curso.demo.repository (Acesso a Dados)

• br.com.curso.demo.security (Configurações de Segurança e Serviço de Autenticação)

• br.com.curso.demo.controller (Controladores Web)

---

1. Configuração do Banco de Dados

Arquivo: application.properties **Local:** src/main/resources

Configuramos a conexão com o MySQL e habilitamos o Hibernate para atualizar as tabelas automaticamente.

spring.application.name=demo-security

Conexão MySQL

spring.datasource.url=jdbc:mysql://localhost:3306/curso_spring_db

spring.datasource.username=root

spring.datasource.password=sua_senha_aqui

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

Thymeleaf (Opcional, para garantir cache desativado em dev)

spring.thymeleaf.cache=false

---

2. A Entidade Usuário (Model)

Arquivo: Usuario.java **Pacote:** br.com.curso.demo.model

Para usar a autenticação via banco de dados (JPA), nossa classe deve implementar a interface UserDetails do Spring Security. Isso permite que o Spring trate nossa classe de domínio como um usuário de sistema.

package br.com.curso.demo.model;

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;

@Entity
public class Usuario implements UserDetails {

    @Id
    private String login; // O login será o ID (chave primária)
    private String nomeCompleto;
    private String senha;

    // Getters e Setters
    public String getLogin() { return login; }
    public void setLogin(String login) { this.login = login; }

    public String getNomeCompleto() { return nomeCompleto; }
    public void setNomeCompleto(String nomeCompleto) { this.nomeCompleto =
nomeCompleto; }

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

    // Métodos da interface UserDetails [2]
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null; // Simplificado: sem perfis de acesso (roles) por enquanto
    }
```

```
    @Override
    public String getPassword() {
        return this.senha; // Retorna a senha para validação
    }

    @Override
    public String getUsername() {
        return this.login; // Retorna o login como "nome de usuário"
    }

    @Override
    public boolean isAccountNonExpired() { return true; }

    @Override
    public boolean isAccountNonLocked() { return true; }

    @Override
    public boolean isCredentialsNonExpired() { return true; }

    @Override
    public boolean isEnabled() { return true; }
}
```

---

3. Repositório de Dados

Arquivo: UsuarioRepository.java **Pacote:** br.com.curso.demo.repository

Interface responsável por buscar o usuário no banco pelo Login.

```
package br.com.curso.demo.repository;

import br.com.curso.demo.model.Usuario;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface UsuarioRepository extends JpaRepository<Usuario, String> {
   Usuario findByLogin(String login); // Método mágico do Spring Data
}
```

---

4. Serviço de Autenticação

Arquivo: AutenticacaoService.java **Pacote:** br.com.curso.demo.security

Esta classe implementa UserDetailsService. Ela é usada pelo Spring Security durante o login para carregar os dados do usuário do banco. Se o usuário não for encontrado, lançamos uma exceção.

```
package br.com.curso.demo.security;

import br.com.curso.demo.model.Usuario;
import br.com.curso.demo.repository.UsuarioRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class AutenticacaoService implements UserDetailsService {

   @Autowired
   private UsuarioRepository usuarioRepository;

   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
      Usuario usuario = usuarioRepository.findByLogin(username);

      if (usuario == null) {
```

```
        throw new UsernameNotFoundException("Usuário não encontrado!");
      }

      return usuario;
    }
}
```

---

5. Configuração de Segurança (Login e Senha)

Arquivo: WebSecurityConfig.java **Pacote:** br.com.curso.demo.security

Aqui definimos as regras de acesso, a página de login personalizada e o codificador de senhas (BCryptPasswordEncoder). Também configuramos o comportamento de logout para invalidar a sessão.

```
package br.com.curso.demo.security;


import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;


@Configuration
@EnableWebSecurity
public class WebSecurityConfig {


  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
      .authorizeHttpRequests((requests) -> requests
        // Permite acesso sem login à página de cadastro e ao salvamento
```

```
            .requestMatchers("/cadastro", "/salvarUsuario", "/css/**").permitAll()
            // Qualquer outra requisição requer autenticação
            .anyRequest().authenticated()
        )
        .formLogin((form) -> form
            .loginPage("/login") // Define nossa página customizada
            .defaultSuccessUrl("/home", true) // Redireciona após sucesso
            .permitAll()
        )
        .logout((logout) -> logout
            .logoutUrl("/logout") // URL para acionar logout
            .logoutSuccessUrl("/login?logout") // Para onde vai após sair
            .permitAll()
        );

    return http.build();
  }

  // Bean para criptografar senhas (Hash) [6]
  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }
}
```

---

6. Controlador Web (Controller)

Arquivo: UsuarioController.java **Pacote:** br.com.curso.demo.controller

Gerencia as requisições HTTP. Observe que ao salvar o usuário, criptografamos a senha antes de enviar ao repositório.

```
package br.com.curso.demo.controller;
```

```
import br.com.curso.demo.model.Usuario;
import br.com.curso.demo.repository.UsuarioRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class UsuarioController {

  @Autowired
  private UsuarioRepository repository;

  @Autowired
  private PasswordEncoder passwordEncoder; // Injeta o codificador

  // Página de Login Personalizada
  @GetMapping("/login")
  public String loginPage() {
    return "login"; // Retorna o arquivo login.html
  }

  // Página Principal (Protegida)
  @GetMapping("/home")
  public String homePage(Model model) {
    model.addAttribute("usuarios", repository.findAll());
    return "home"; // Retorna home.html
  }
```

```
    // Formulário de Cadastro de Novo Usuário
    @GetMapping("/cadastro")
    public String cadastroPage() {
        return "cadastro"; // Retorna cadastro.html
    }

    // Ação de Salvar Usuário
    @PostMapping("/salvarUsuario")
    public String salvarUsuario(Usuario usuario) {
        // Criptografa a senha antes de salvar [6]
        String senhaCriptografada = passwordEncoder.encode(usuario.getSenha());
        usuario.setSenha(senhaCriptografada);

        repository.save(usuario);
        return "redirect:/login"; // Redireciona para o login após cadastrar
    }
}
```

---

7. Páginas HTML (Thymeleaf)

Local: src/main/resources/templates

O Thymeleaf processa estes arquivos no servidor antes de enviar ao navegador.

7.1. Login Personalizado

Arquivo: login.html

O Spring Security espera que os campos se chamem username e password por padrão.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Login do Sistema</title>
</head>
<body>
    <h2>Acesso ao Sistema</h2>
```

```
  <!-- Mensagem de erro ao falhar login -->
  <div th:if="${param.error}" style="color:red">
    Usuário ou senha inválidos.
  </div>
  <!-- Mensagem de logout -->
  <div th:if="${param.logout}" style="color:green">
    Você saiu do sistema.
  </div>

  <!-- O action deve apontar para a mesma URL configurada no loginPage, method POST --
>
  <form th:action="@{/login}" method="post">
    <div>
      <label>Login:</label>
      <input type="text" name="username" placeholder="Seu login" required/>
    </div>
    <div>
      <label>Senha:</label>
      <input type="password" name="password" placeholder="Sua senha" required/>
    </div>
    <button type="submit">Entrar</button>
  </form>

  <br/>
  <a href="/cadastro">Criar nova conta</a>
</body>
</html>
```

7.2. Cadastro de Usuário

Arquivo: cadastro.html

Formulário público para criar novos usuários no banco.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Cadastro de Usuário</title>
</head>
<body>
  <h2>Novo Usuário</h2>

  <form th:action="@{/salvarUsuario}" method="post">
    <div>
      <label>Login (ID):</label>
      <input type="text" name="login" required/>
    </div>
    <div>
      <label>Nome Completo:</label>
      <input type="text" name="nomeCompleto" required/>
    </div>
    <div>
      <label>Senha:</label>
      <input type="password" name="senha" required/>
    </div>
    <button type="submit">Cadastrar</button>
  </form>

  <a href="/login">Voltar para Login</a>
</body>
</html>
```

7.3. Página Principal (Home) com Logout

Arquivo: home.html

Esta página só é acessível se o usuário estiver logado (sessão válida). Aqui listamos todos os usuários cadastrados e oferecemos o botão de Logoff.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Painel Principal</title>
</head>
<body>
  <h1>Bem-vindo ao Sistema!</h1>

  <h3>Usuários Cadastrados:</h3>
  <table border="1">
    <thead>
      <tr>
        <th>Login</th>
        <th>Nome</th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="u : ${usuarios}">
        <td th:text="${u.login}">login</td>
        <td th:text="${u.nomeCompleto}">nome</td>
      </tr>
    </tbody>
  </table>

  <br/><br/>

  <!-- Botão de Logout -->
  <!-- Por segurança, logout deve ser um POST em configurações padrão -->
  <form th:action="@{/logout}" method="post">
    <button type="submit" style="background-color: salmon;">Sair do Sistema
(Logoff)</button>
```

```
  </form>


</body>

</html>
```

Resumo do Fluxo de Execução

1. Ao iniciar a aplicação, acesse http://localhost:8080/home.

2. O **Filtro de Segurança** interceptará a requisição e redirecionará para /login (pois você não tem sessão).

3. Clique em "Criar nova conta". Preencha o formulário e envie. O Controller salvará a senha criptografada com **BCrypt**.

4. Faça login com as credenciais criadas. O AutenticacaoService buscará o usuário no banco.

5. Ao logar, uma **Sessão** (JSESSIONID) é criada.

6. Ao clicar em "Sair do Sistema", a sessão é invalidada e você retorna à tela de login.