Entendendo Spring MVC e Segurança

Esta leitura fornece uma introdução abrangente à construção de aplicações web usando Spring MVC e à sua segurança com Spring Security. Compreender esses conceitos ajudará você a configurar um projeto Spring MVC, integrar templates, manipular formulários e proteger aplicações por meio de autenticação e autorização. Vamos explorar os seguintes conceitos de codificação em Java:

- Introdução ao Spring MVC
- Templating usando Thymeleaf
- Manipulação de formulários usando Spring MVC
- Visão geral do Spring Security
- Autenticação e Autorização
- Segurança baseada em formulários usando Spring Security

Mantenha este resumo disponível como referência à medida que você avança em seu curso e consulte esta leitura ao começar a codificar em Java após este curso!

Introdução ao Spring MVC

Spring Boot simplifica a configuração e a configuração de um projeto Spring MVC.

Descrição	**Exemplo**
Código de exemplo para um Modelo	`public class Book {` `  private String title;` `  private String author;` `  private double price;` `  // Getters e setters` `}`
Código de exemplo para uma Visão	`<!-- bookstore.html -->` `<div class="book-details">` `  <h2>${book.title}</h2>` `  <p>Autor: ${book.author}</p>` `  <p>Preço: $${book.price}</p>` `</div>`
Código de exemplo para um Controlador	`@Controller` `public class BookController {` `  @GetMapping("/books/{id}")` `  public String getBook(@PathVariable Long id, Model model) {` `  Book book = bookService.findById(id);` `  model.addAttribute("book", book);` `  return "bookstore";` `  }` `}`

Descrição	Exemplo
Criando um Controlador de Pesquisa	`@Controller` `public class BookSearchController {` `  @GetMapping("/books/search")` `  public String searchBooks(@RequestParam String title, Model model) {` `  List<Book> books = bookService.searchByTitle(title);` `  model.addAttribute("searchResults", books);` `  return "searchResults";` `  }` `}`

Introdução ao Spring MVC usando Spring Boot

Um aplicativo web Spring Boot pode ser configurado implementando as camadas de Modelo, Controlador e Visão.

Descrição	Exemplo
Configurando dependências a serem incluídas no arquivo pom.xml	`<dependencies>` `  <!-- Spring Boot Web Starter - Inclui Spring MVC e Tomcat embutido -->` `  <dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-starter-web</artifactId>` `  </dependency>` `  <!-- Tomcat Jasper - Necessário para processamento de JSP -->` `  <dependency>` `  <groupId>org.apache.tomcat.embed</groupId>` `  <artifactId>tomcat-embed-jasper</artifactId>` `  <scope>provided</scope>` `  </dependency>` `  <!-- JSTL - JavaServer Pages Standard Tag Library -->` `  <dependency>` `  <groupId>javax.servlet</groupId>` `  <artifactId>jstl</artifactId>` `  </dependency>` `  <!-- Spring Boot DevTools - Habilita reinício automático -->` `  <dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-devtools</artifactId>` `  <scope>runtime</scope>` `  <optional>true</optional>` `  </dependency>` `</dependencies>`

Descrição	Exemplo
Criando application.properties em src/main/resources	`# Configuração do resolutor de visualização para JSP` `spring.mvc.view.prefix=/WEB-INF/views/` `spring.mvc.view.suffix=.jsp` `# Configuração do servidor` `server.port=8080` `# Ferramentas de desenvolvimento` `spring.devtools.restart.enabled=true`
Criando a camada de Modelo	`package com.example.model;` `public class Employee {` `  // Campos privados para encapsulamento` `  private int id;` `  private String name;` `  private String department;` `  private double salary;` `  private String email;` `  // Construtor padrão - necessário para vinculação de formulário` `  public Employee() {` `  }` `  // Construtor parametrizado para criar objetos de funcionário` `  public Employee(int id, String name, String department, double salary, String email) {` `  this.id = id;` `  this.name = name;` `  this.department = department;` `  this.salary = salary;` `  this.email = email;` `  }` `  // Getters e Setters com validação` `  public int getId() {` `  return id;` `  }` `  public void setId(int id) {` `  if (id > 0) {` `  this.id = id;` `  } else {` `  throw new IllegalArgumentException("ID deve ser positivo");` `  }` `  }` `  // Getters e setters adicionais com validação semelhante` `  // ... (incluir todos os getters e setters)` `  // método toString para depuração e registro` `  @Override` `  public String toString() {` `  return "Employee{" +` `  "id=" + id +` `  ", name='" + name + ''' +` `  ", department='" + department + ''' +` `  ", salary=" + salary +` `  ", email='" + email + ''' +` `  '}';` `  }` `}`
Criando a camada de Controlador	`package com.example.controller;` `import com.example.model.Employee;` `import org.springframework.stereotype.Controller;` `import org.springframework.ui.Model;` `import org.springframework.web.bind.annotation.*;` `import java.util.ArrayList;` `import java.util.List;`

Descrição	Exemplo
	`` @Controller @RequestMapping("/employees") // URL base para todas as operações de funcionário public class EmployeeController { // Lista em memória para armazenar funcionários (substituir por banco de dados em produção) private List<Employee> employees = new ArrayList<>(); private int nextId = 1; // Gerador de ID simples // Construtor para inicializar dados de exemplo public EmployeeController() { // Adicionar funcionários de exemplo para demonstração addSampleEmployees(); } // Manipulador para exibir todos os funcionários @GetMapping public String listEmployees(Model model) { // Adicionar lista de funcionários ao modelo para renderização da visão model.addAttribute("employees", employees); // Retornar nome da visão - será resolvido para /WEB-INF/views/employees/list.jsp return "employees/list"; } // Manipulador para exibir o formulário de adicionar funcionário @GetMapping("/add") public String showAddForm(Model model) { // Adicionar objeto funcionário vazio ao modelo para vinculação de formulário model.addAttribute("employee", new Employee()); return "employees/addForm"; } // Manipulador para processar a submissão do formulário de adicionar funcionário @PostMapping("/add") public String addEmployee(@ModelAttribute Employee employee) { // Definir o próximo ID disponível employee.setId(nextId++); // Adicionar à nossa lista employees.add(employee); // Redirecionar para evitar reenvio do formulário return "redirect:/employees"; } // Manipulador para exibir detalhes do funcionário @GetMapping("/{id}") public String viewEmployee(@PathVariable int id, Model model) { // Encontrar funcionário pelo ID Employee employee = findEmployeeById(id); if (employee != null) { model.addAttribute("employee", employee); return "employees/view"; } return "redirect:/employees"; } // Método auxiliar para encontrar funcionário pelo ID private Employee findEmployeeById(int id) { return employees.stream() .filter(emp -> emp.getId() == id) .findFirst() .orElse(null); } // Método auxiliar para adicionar dados de exemplo private void addSampleEmployees() { employees.add(new Employee(nextId++, "John Doe", "IT", 75000.0, "john@example.com")); employees.add(new Employee(nextId++, "Jane Smith", "HR", 65000.0, "jane@example.com")); } } ```
Criando a Visualização da lista	``` <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <!DOCTYPE html> <html> <head> <title>Gerenciamento de Funcionários - Lista</title> <style> /* Adicionar um estilo básico */ table { width: 100%; border-collapse: collapse; margin: 20px 0; } th, td { padding: 10px; ```

Descrição	Exemplo
	border: 1px solid #ddd; text-align: left; } th { background-color: #f5f5f5; } .action-links { margin: 20px 0; } </style> </head> <body> <h1>Diretório de Funcionários</h1> Adicionar Novo Funcionário </div> <table> <thead> <tr> <th>ID</th> <th>Nome</th> <th>Departamento</th> <th>Salário</th> <th>Email</th> <th>Ações</th> </tr> </thead> <tbody> <c:forEach items="${employees}" var="emp"> <tr> <td>${emp.id}</td> <td>${emp.name}</td> <td>${emp.department}</td> <td>$${emp.salary}</td> <td> Ver </td> </tr> </c:forEach> </tbody> </table> </body> </html>
Criando um formulário de Adição	<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%> <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %> <!DOCTYPE html> <html> <head> <title>Adicionar Novo Funcionário</title> <style> .form-group { margin-bottom: 15px; } label { display: block; margin-bottom: 5px; } input[type="text"], input[type="email"], input[type="number"] { width: 300px; padding: 8px; border: 1px solid #ddd; border-radius: 4px; } button { padding: 10px 20px; background-color: #007bff; color: white; border: none; border-radius: 4px; cursor: pointer; } </style> </head> <body> Adicionar Novo Funcionário <form:form action="${pageContext.request.contextPath}/employees/add" method="post" modelAttribute="employee">

Descrição	Exemplo
	 <label for="name">Nome:</label> <form:input path="name" required="true" /> <label for="department">Departamento:</label> <form:input path="department" required="true" /> <label for="salary">Salário:</label> <form:input path="salary" type="number" step="0.01" required="true" /> <label for="email">Email:</label> <form:input path="email" type="email" required="true" /> <button type="submit">Adicionar Funcionário</button> </form:form> Voltar para Lista </body> </html>

Modelagem usando Thymeleaf

Para a geração de conteúdo web dinâmico, você pode configurar o Thymeleaf em um projeto Spring Boot.

Descrição	Exemplo
Configurando o Thymeleaf em um projeto Spring Boot	<dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-thymeleaf</artifactId> </dependency>
Criando um template Thymeleaf, books.html, para exibir a lista de livros	<!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Lista de Livros</title> </head> <body> Livros Disponíveis <!-- Iterando sobre uma lista de livros --> <li th:each="book : ${books}"> <!-- Exibindo o título do livro --> Título do Livro por <!-- Exibindo o nome do autor --> Nome do Autor - Preço: $0.00 </body> </html>

Descrição	Exemplo
Criando um Controlador Spring MVC para lidar com requisições e exibir a lista	`import org.springframework.stereotype.Controller;` `import org.springframework.ui.Model;` `import org.springframework.web.bind.annotation.GetMapping;` `import java.util.List;` `import java.util.ArrayList;` `@Controller` `public class BookController {` `  @GetMapping("/books")` `  public String getBooks(Model model) {` `  // Criando uma lista de livros` `  List<Book> books = new ArrayList<>();` `  books.add(new Book("Effective Java", "Joshua Bloch", 45.00));` `  books.add(new Book("Spring in Action", "Craig Walls", 40.00));` `  books.add(new Book("Clean Code", "Robert C. Martin", 50.00));` `  // Adicionando a lista de livros ao modelo` `  model.addAttribute("books", books);` `  // Retornando o nome do arquivo do template (sem a extensão)` `  return "books";` `  }` `}`
Criando uma Classe Book com getters e setters para cada propriedade	`public class Book {` `  private String title;` `  private String author;` `  private double price;` `  public Book(String title, String author, double price) {` `  this.title = title;` `  this.author = author;` `  this.price = price;` `  }` `  // Getter para o título` `  public String getTitle() {` `  return title;` `  }` `  // Setter para o título` `  public void setTitle(String title) {` `  this.title = title;` `  }` `  // Getter para o autor` `  public String getAuthor() {` `  return author;` `  }` `  // Setter para o autor` `  public void setAuthor(String author) {` `  this.author = author;` `  }` `  // Getter para o preço` `  public double getPrice() {` `  return price;` `  }` `  // Setter para o preço` `  public void setPrice(double price) {` `  this.price = price;` `  }` `}`

Descrição	Exemplo
Criando um arquivo CSS externo, styles.css, no diretório src/main/resources/static/css	`body { font-family: Arial, sans-serif; background-color: #f4f4f9; color: #333; margin: 0; padding: 20px; } h1 { color: #4a90e2; } ul { list-style-type: none; padding: 0; } li { background-color: #fff; margin-bottom: 10px; padding: 15px; border-radius: 5px; box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); } span { display: inline-block; margin-right: 10px; } .price { font-weight: bold; color: #e94e77; }`
Atualizando o template books.html para incluir o arquivo CSS e adicionar alguns estilos inline usando Thymeleaf	`<!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Lista de Livros</title> <!-- Link para o arquivo CSS externo --> <link rel="stylesheet" th:href="@{/css/styles.css}" /> </head> <body> Livros Disponíveis <!-- Iterando sobre uma lista de livros --> <li th:each="book : ${books}" th:style="'background-color:' + (${book.price} > 45 ? '#eef9f9' : '#fff')"> <!-- Exibindo o título do livro --> 15 ? 'bold' : 'normal')"> Título do Livro por <!-- Exibindo o nome do autor --> Nome do Autor - Preço: $0.00 </body> </html>`

Manipulação de formulários usando Spring MVC

Descrição	Exemplo
Criando uma	`package com.example.demo.model;`

Descrição	Exemplo
Classe Modelo User	`public class User {` `  private String firstName;` `  private String lastName;` `  private String email;` `  // Getters e Setters` `  public String getFirstName() {` `  return firstName;` `  }` `  public void setFirstName(String firstName) {` `  this.firstName = firstName;` `  }` `  public String getLastName() {` `  return lastName;` `  }` `  public void setLastName(String lastName) {` `  this.lastName = lastName;` `  }` `  public String getEmail() {` `  return email;` `  }` `  public void setEmail(String email) {` `  this.email = email;` `  }` `}`
Criando um Controlador para lidar com requisições HTTP	`package com.example.demo.controller;` `import com.example.demo.model.User;` `import org.springframework.stereotype.Controller;` `import org.springframework.ui.Model;` `import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.PostMapping;` `import org.springframework.web.bind.annotation.ModelAttribute;` `@Controller` `public class UserController {` `  // Exibir o formulário para o usuário` `  @GetMapping("/userForm")` `  public String showForm(Model model) {` `  model.addAttribute("user", new User());` `  return "userForm";` `  }` `  // Lidar com a submissão do formulário` `  @PostMapping("/submitForm")` `  public String submitForm(@ModelAttribute User user, Model model) {` `  model.addAttribute("user", user);` `  return "formResult";` `  }` `}`
Criando um template de View para exibir o formulário	`userForm.html` `<!DOCTYPE html>` `<html xmlns:th="http://www.thymeleaf.org">` `<head>` `  <title>Formulário do Usuário</title>` `</head>` `<body>` `<h2>Digite os Detalhes do Usuário</h2>` `<form th:action="@{/submitForm}" th:object="${user}" method="post">` `  <label>Nome:</label>` `  <input type="text" th:field="*{firstName}" /> ` `  <label>Sobrenome:</label>` `  <input type="text" th:field="{lastName}" /> `

Descrição	Exemplo
	<label>Email:</label> <input type="email" th:field="*{email}" /> <button type="submit">Enviar</button> </form> </body> </html>
Criando um template de View para mostrar os resultados	formResult.html <!DOCTYPE html> <html xmlns:th="http://www.thymeleaf.org"> <head> <title>Resultado do Formulário</title> </head> <body> Detalhes do Usuário Submetidos Nome: Sobrenome: <span th:text="${user.lastName}"> Email: </body> </html>
Adicionando anotações de validação ao Modelo para especificar regras de validação para cada campo	package com.example.demo.model; import jakarta.validation.constraints.Email; import jakarta.validation.constraints.NotEmpty; import jakarta.validation.constraints.Size; public class User { @NotEmpty(message = "O nome é obrigatório") @Size(min = 2, max = 30, message = "O nome deve ter entre 2 e 30 caracteres") private String firstName; @NotEmpty(message = "O sobrenome é obrigatório") @Size(min = 2, max = 30, message = "O sobrenome deve ter entre 2 e 30 caracteres") private String lastName; @NotEmpty(message = "O email é obrigatório") @Email(message = "Por favor, forneça um endereço de email válido") private String email; // Getters e Setters public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } public String getLastName() { return lastName; } public void setLastName(String lastName) { this.lastName = lastName; } public String getEmail() { return email; } public void setEmail(String email) { this.email = email; } }

Descrição	Exemplo
Atualizando o Controlador para lidar com erros de validação	`package com.example.demo.controller;` `import com.example.demo.model.User;` `import jakarta.validation.Valid;` `import org.springframework.stereotype.Controller;` `import org.springframework.ui.Model;` `import org.springframework.validation.BindingResult;` `import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.ModelAttribute;` `import org.springframework.web.bind.annotation.PostMapping;` `@Controller` `public class UserController {` `    @GetMapping("/userForm")` `    public String showForm(Model model) {` `        model.addAttribute("user", new User());` `        return "userForm";` `    }` `    @PostMapping("/submitForm")` `    public String submitForm(@Valid @ModelAttribute User user, BindingResult bindingResult, Model model) {` `        if (bindingResult.hasErrors()) {` `            return "userForm";` `        }` `        model.addAttribute("user", user);` `        return "formResult";` `    }` `}`
Atualizando a View do Formulário para exibir mensagens de erro de validação	`<!DOCTYPE html>` `<html xmlns:th="http://www.thymeleaf.org">` `<head>` `    <title>Formulário do Usuário</title>` `</head>` `<body>` `<h2>Digite os Detalhes do Usuário</h2>` `<form th:action="@{/submitForm}" th:object="${user}" method="post">` `    <div>` `        <label>Nome:</label>` `        <input type="text" th:field="*{firstName}" />` `        <div th:if="${#fields.hasErrors('firstName')}" th:errors="{firstName}"></div>` `    </div>` `    <div>` `        <label>Sobrenome:</label>` `        <input type="text" th:field="*{lastName}" />` `        <div th:if="${#fields.hasErrors('lastName')}" th:errors="{lastName}"></div>` `    </div>` `    <div>` `        <label>Email:</label>` `        <input type="email" th:field="*{email}" />` `        <div th:if="${#fields.hasErrors('email')}" th:errors="{email}"></div>` `    </div>` `    <button type="submit">Enviar</button>` `</form>` `</body>` `</html>`

Autenticação e autorização

O processo de implementar autenticação e autorização em uma aplicação Spring Boot é simples.

Descrição	Exemplo
Incluindo Spring Security no projeto	`<dependency>` `  <groupId>org.springframework.boot</groupId>` `  <artifactId>spring-boot-starter-security</artifactId>` `</dependency>`
Criando uma Classe de Configuração de Segurança	`import org.springframework.context.annotation.Configuration;` `import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;` `import org.springframework.security.config.annotation.web.builders.HttpSecurity;` `import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;` `import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;` `@Configuration` `@EnableWebSecurity` `public class WebSecurityConfig extends WebSecurityConfigurerAdapter {` `  @Override` `  protected void configure(AuthenticationManagerBuilder auth) throws Exception {` `  auth.inMemoryAuthentication()` `  .withUser("user").password("{noop}password").roles("USER")` `  .and()` `  .withUser("admin").password("{noop}adminpass").roles("ADMIN");` `  }` `  @Override` `  protected void configure(HttpSecurity http) throws Exception {` `  http` `  .authorizeRequests()` `  .antMatchers("/admin/**").hasRole("ADMIN")` `  .antMatchers("/user/**").hasRole("USER")` `  .antMatchers("/public/**").permitAll()` `  .and()` `  .formLogin();` `  }` `}`
Adicionando endpoints de Controlador	`import org.springframework.web.bind.annotation.GetMapping;` `import org.springframework.web.bind.annotation.RestController;` `@RestController` `public class SampleController {` `  @GetMapping("/public/welcome")` `  public String welcome() {` `  return "Bem-vindo ao endpoint público!";` `  }` `  @GetMapping("/user/profile")` `  public String userProfile() {` `  return "Perfil do Usuário: Acesso Permitido";` `  }` `  @GetMapping("/admin/dashboard")` `  public String adminDashboard() {` `  return "Painel do Admin: Acesso Permitido";` `  }` `}`

Descrição	Exemplo
Criando um endpoint acessível publicamente	`@GetMapping("/public/welcome")` `public String welcome() {` `  return "Bem-vindo ao endpoint público!";` `}`
Criando um endpoint acessível apenas a usuários específicos	`@GetMapping("/user/profile")` `@PreAuthorize("hasRole('USER')")` `public String userProfile() {` `  return "Perfil do Usuário: Acesso Permitido";` `}`
Criando um endpoint somente para administradores	`@GetMapping("/admin/dashboard")` `@PreAuthorize("hasRole('ADMIN')")` `public String adminDashboard() {` `  return "Painel do Admin: Acesso Permitido";` `}`

Segurança baseada em formulário usando Spring Security

As configurações do Spring Security permitem restringir o acesso e habilitar a autenticação baseada em formulário.

Descrição	Exemplo
Criando uma classe SecurityConfig em src/main/java/com/example/demosecurity/config	`package com.example.demosecurity.config;` `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.config.annotation.web.configuration.WebSecurityConfigurerAdapter;` `@Configuration` `@EnableWebSecurity` `public class SecurityConfig extends WebSecurityConfigurerAdapter {` `  @Override` `  protected void configure(HttpSecurity http) throws Exception {` `  http` `  .authorizeRequests()` `  .antMatchers("/public/**").permitAll() // Permitir acesso a URLs públicas` `  .anyRequest().authenticated() // Exigir autenticação para outras URLs` `  .and()` `  .formLogin() // Habilitar autenticação baseada em formulário` `  .loginPage("/login") // URL da página de login personalizada` `  .permitAll()` `  .and()` `  .logout() // Habilitar suporte a logout` `  .permitAll();` `  }` `}`

Descrição	Exemplo
Criando uma página de login personalizada	`<!DOCTYPE html>` `<html>` `<head>` `<title>Login</title>` `</head>` `<body>` `<h2>Login</h2>` `<form method="post" action="/login">` `<div>` `<label>Nome de usuário:</label>` `<input type="text" name="username">` `</div>` `<div>` `<label>Senha:</label>` `<input type="password" name="password">` `</div>` `<div>` `<button type="submit">Login</button>` `</div>` `</form>` `</body>` `</html>`
Incluindo a dependência do Spring Security em pom.xml para Maven	`<dependency>` `<groupId>org.springframework.boot</groupId>` `<artifactId>spring-boot-starter-security</artifactId>` `</dependency>`
Configurando o Codificador de Senhas	`package com.example.demosecurity.config;` `import org.springframework.context.annotation.Bean;` `import org.springframework.context.annotation.Configuration;` `import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;` `import org.springframework.security.crypto.password.PasswordEncoder;` `@Configuration` `public class SecurityConfig {` `@Bean` `public PasswordEncoder passwordEncoder() {` `return new BCryptPasswordEncoder();` `}` `}`
Criptografando senhas antes de salvar	`package com.example.demosecurity.service;` `import com.example.demosecurity.model.User;` `import com.example.demosecurity.repository.UserRepository;` `import org.springframework.beans.factory.annotation.Autowired;` `import org.springframework.security.crypto.password.PasswordEncoder;` `import org.springframework.stereotype.Service;` `@Service`

Descrição	Exemplo
	`public class UserService {` `@Autowired` `private UserRepository userRepository;` `@Autowired` `private PasswordEncoder passwordEncoder;` `public void saveUser(User user) {` `// Criptografar a senha do usuário antes de salvar` `user.setPassword(passwordEncoder.encode(user.getPassword()));` `userRepository.save(user);` `}` `}`
Criando uma entidade de usuário simples	`package com.example.demosecurity.model;` `import javax.persistence.Entity;` `import javax.persistence.GeneratedValue;` `import javax.persistence.GenerationType;` `import javax.persistence.Id;` `@Entity` `public class User {` `@Id` `@GeneratedValue(strategy = GenerationType.IDENTITY)` `private Long id;` `private String username;` `private String password;` `// Getters e setters` `public Long getId() {` `return id;` `}` `public void setId(Long id) {` `this.id = id;` `}` `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;` `}` `}`

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan