**Documentação Geral Projeto Final**

**Curso:** Técnico em Desenvolvimento de Sistemas

**Instituição:** Senai

Apresento um passo a passo completo e detalhado para rodar este sistema de agendamento de serviços de estética automotiva (com módulos para clientes, agendamentos, produtos, vendas, serviços e funcionários) usando Java 21, Spring Boot, Thymeleaf e MySQL. Esse tutorial vai permitir que você rode o servidor na sua máquina e acesse o sistema via navegador (por exemplo, pelo link http://localhost:8080), podendo ser acessado de qualquer dispositivo que esteja na mesma rede (ou via domínio/IP, se configurado).

**1. Pré-requisitos**

**Java 21:** Certifique-se de ter o JDK 21 instalado.

**Maven:** Usaremos Maven para gerenciamento de dependências e build.

**Banco de dados:** MySQL Workbench.

**IDE:** VS Code.

**2. Criação do Projeto com Springboot**

**2.1. Gerando o Projeto**

Você pode criar um novo projeto Spring Boot através do Spring Initializr com as seguintes configurações:

**Project:** Maven Project

**Language:** Java

**Spring Boot:** Use a versão mais recente que suporte Java 21 (por exemplo, 3.4.3).

**Group:** com.erp

Artifact: estetica

Packaging: Jar

Java: 21

**2.2. Dependências**

Adicione as seguintes dependências:

Spring Web

Spring Data JPA

Thymeleaf

MySQL Driver

(Opcional) Spring Boot DevTools (para reinício automático)

O arquivo pom.xml terá algo parecido com:

**<project ...>  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.seuprojeto</groupId>  
 <artifactId>estetica-auto</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <properties>  
 <java.version>21</java.version>  
 </properties>  
 <dependencies>  
 <!-- Dependências do Spring Boot -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <!-- Spring Data JPA -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <!-- Thymeleaf -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <!-- MySQL Driver -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <!-- DevTools (opcional) -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>**

**3. Configuração do Banco de Dados**

No arquivo src/main/resources/application.properties (ou application.yml se preferir YAML), **configure a conexão com o MySQL:**

**spring.datasource.url=jdbc:mysql://localhost:3306/estetica\_auto?useSSL=false&serverTimezone=UTC  
spring.datasource.username=SEU\_USUARIO  
spring.datasource.password=SUA\_SENHA  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver**  
  
**# Como o banco já existe e está populado, não queremos que o JPA modifique o esquema:  
spring.jpa.hibernate.ddl-auto=none**  
  
**# Exibir as queries no console (opcional)  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true**

**Atenção: Substitua SEU\_USUARIO e SUA\_SENHA pelas credenciais reais do seu MySQL.**

**4. Criação das Entidades (Model)**

**Crie um pacote, por exemplo, com.erp.estetica.model e crie uma classe para cada tabela do banco.**

**4.1. Cliente**

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "clientes")  
public class Cliente {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 private String nome;  
 private String telefone;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNome() {  
 return nome;  
 }  
 public void setNome(String nome) {  
 this.nome = nome;  
 }  
 public String getTelefone() {  
 return telefone;  
 }  
 public void setTelefone(String telefone) {  
 this.telefone = telefone;  
 }  
}**

**4.2. Produto**

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
import java.math.BigDecimal;  
  
@Entity  
@Table(name = "produtos")  
public class Produto {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 private String nome;  
 private BigDecimal preco;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNome() {  
 return nome;  
 }  
 public void setNome(String nome) {  
 this.nome = nome;  
 }  
 public BigDecimal getPreco() {  
 return preco;  
 }  
 public void setPreco(BigDecimal preco) {  
 this.preco = preco;  
 }  
}**

**4.3. Venda**

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "vendas")  
public class Venda {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 @ManyToOne  
 @JoinColumn(name = "produto\_id")  
 private Produto produto;  
  
 @ManyToOne  
 @JoinColumn(name = "cliente\_id")  
 private Cliente cliente;  
  
 private int quantidade;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public Produto getProduto() {  
 return produto;  
 }  
 public void setProduto(Produto produto) {  
 this.produto = produto;  
 }  
 public Cliente getCliente() {  
 return cliente;  
 }  
 public void setCliente(Cliente cliente) {  
 this.cliente = cliente;  
 }  
 public int getQuantidade() {  
 return quantidade;  
 }  
 public void setQuantidade(int quantidade) {  
 this.quantidade = quantidade;  
 }  
}**

**4.4. Serviço**

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
import java.math.BigDecimal;  
  
@Entity  
@Table(name = "servicos")  
public class Servico {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 private String nome;  
 private BigDecimal preco;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNome() {  
 return nome;  
 }  
 public void setNome(String nome) {  
 this.nome = nome;  
 }  
 public BigDecimal getPreco() {  
 return preco;  
 }  
 public void setPreco(BigDecimal preco) {  
 this.preco = preco;  
 }  
}**

**4.5. Agendamento**

**Na tabela, o campo cliente\_nome é um texto. Se desejar relacionar com a tabela de clientes, poderá ser alterado;**

**O campo cliente\_nome é do tipo texto (VARCHAR ou TEXT), mas se você deseja relacioná-lo corretamente com a tabela de clientes, o ideal é armazenar o ID do cliente em vez do nome. Aqui está o motivo:**

### 1. ****Chave Estrangeira e Relacionamento Correto****

* Em um banco de dados relacional, a melhor prática é utilizar **chaves primárias e estrangeiras** para estabelecer relações entre tabelas.
* O nome do cliente pode se repetir, mas o **ID do cliente (geralmente um número, como INT ou BIGINT)** será único.

### 2. ****Otimização de Performance****

* Comparar números (INT, BIGINT) é **mais rápido e eficiente** do que comparar textos (VARCHAR ou TEXT).
* Índices em colunas numéricas ocupam menos espaço e são mais eficientes do que em colunas de texto.

### 3. ****Facilidade na Manutenção e Integridade dos Dados****

* Se o nome do cliente mudar, você teria que atualizar **todas as tabelas** que armazenam o nome.
* Se usar o **ID do cliente**, basta atualizar o nome na tabela de clientes, sem afetar outras tabelas.

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
import java.time.LocalDateTime;  
  
@Entity  
@Table(name = "agendamentos")  
public class Agendamento {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 @ManyToOne  
 @JoinColumn(name = "servico\_id")  
 private Servico servico;  
  
 @Column(name = "cliente\_nome")  
 private String clienteNome;  
  
 @Column(name = "data\_hora")  
 private LocalDateTime dataHora;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public Servico getServico() {  
 return servico;  
 }  
 public void setServico(Servico servico) {  
 this.servico = servico;  
 }  
 public String getClienteNome() {  
 return clienteNome;  
 }  
 public void setClienteNome(String clienteNome) {  
 this.clienteNome = clienteNome;  
 }  
 public LocalDateTime getDataHora() {  
 return dataHora;  
 }  
 public void setDataHora(LocalDateTime dataHora) {  
 this.dataHora = dataHora;  
 }  
}**

**4.6. Funcionário**

**package com.seuprojeto.esteticaauto.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "funcionarios")  
public class Funcionario {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 private String nome;  
 private String cargo;  
 private String telefone;  
  
 // Getters e Setters  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNome() {  
 return nome;  
 }  
 public void setNome(String nome) {  
 this.nome = nome;  
 }  
 public String getCargo() {  
 return cargo;  
 }  
 public void setCargo(String cargo) {  
 this.cargo = cargo;  
 }  
 public String getTelefone() {  
 return telefone;  
 }  
 public void setTelefone(String telefone) {  
 this.telefone = telefone;  
 }  
}**

**5. Criação dos Repositórios (DAO)**

**Crie um pacote, por exemplo, com.seuprojeto.esteticaauto.repository e defina interfaces para cada entidade, estendendo JpaRepository.**

Exemplos:

**package com.seuprojeto.esteticaauto.repository;  
  
import com.seuprojeto.esteticaauto.model.Cliente;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface ClienteRepository extends JpaRepository<Cliente, Integer> {  
}**

**package com.seuprojeto.esteticaauto.repository;  
  
import com.seuprojeto.esteticaauto.model.Produto;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface ProdutoRepository extends JpaRepository<Produto, Integer> {  
}**

**Repita o mesmo para Venda, Servico, Agendamento e Funcionario.**

**6. Criação dos Controllers**

**Crie um pacote, por exemplo, com.seuprojeto.esteticaauto.controller. Para cada módulo, crie um controller simples que trate as operações CRUD e a exibição via páginas Thymeleaf.**

**package com.seuprojeto.esteticaauto.controller;  
  
import com.seuprojeto.esteticaauto.model.Agendamento;  
import com.seuprojeto.esteticaauto.model.Servico;  
import com.seuprojeto.esteticaauto.repository.AgendamentoRepository;  
import com.seuprojeto.esteticaauto.repository.ServicoRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.List;  
  
@Controller  
@RequestMapping("/agendamentos")  
public class AgendamentoController {  
  
 @Autowired  
 private AgendamentoRepository agendamentoRepository;  
  
 @Autowired  
 private ServicoRepository servicoRepository;  
  
 // Lista todos os agendamentos  
 @GetMapping  
 public String listar(Model model) {  
 List<Agendamento> agendamentos = agendamentoRepository.findAll();  
 model.addAttribute("agendamentos", agendamentos);  
 return "agendamentos/lista";  
 }  
  
 // Exibe o formulário para criar um novo agendamento  
 @GetMapping("/novo")  
 public String novo(Model model) {  
 List<Servico> servicos = servicoRepository.findAll();  
 model.addAttribute("servicos", servicos);  
 model.addAttribute("agendamento", new Agendamento());  
 return "agendamentos/form";  
 }  
  
 // Salva o agendamento  
 @PostMapping("/salvar")  
 public String salvar(@ModelAttribute Agendamento agendamento, @RequestParam("dataHoraStr") String dataHoraStr) {  
 // Converte a string para LocalDateTime (formato: yyyy-MM-dd'T'HH:mm)  
 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm");  
 agendamento.setDataHora(LocalDateTime.parse(dataHoraStr, formatter));  
 agendamentoRepository.save(agendamento);  
 return "redirect:/agendamentos";  
 }  
  
 // Exibe o formulário para editar um agendamento  
 @GetMapping("/editar/{id}")  
 public String editar(@PathVariable("id") Integer id, Model model) {  
 Agendamento agendamento = agendamentoRepository.findById(id).orElse(null);  
 List<Servico> servicos = servicoRepository.findAll();  
 model.addAttribute("servicos", servicos);  
 model.addAttribute("agendamento", agendamento);  
 return "agendamentos/form";  
 }  
  
 // Remove um agendamento  
 @GetMapping("/deletar/{id}")  
 public String deletar(@PathVariable("id") Integer id) {  
 agendamentoRepository.deleteById(id);  
 return "redirect:/agendamentos";  
 }  
}**

**Observação: Crie controllers similares para clientes, produtos, vendas e funcionários. Cada controller deverá ter métodos para listar, criar, editar e deletar os registros.**

**7. Criação das Páginas Thymeleaf**

**No diretório src/main/resources/templates, crie as páginas para as visualizações.**

**7.1. Página Principal (index.html)**

**<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Sistema Estética Automotiva</title>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">  
</head>  
<body>  
 <div class="container mt-4">  
 <h1>Sistema de Agendamento e Gestão</h1>  
 <ul class="list-group">  
 <li class="list-group-item"><a th:href="@{/clientes}">Clientes</a></li>  
 <li class="list-group-item"><a th:href="@{/funcionarios}">Funcionários</a></li>  
 <li class="list-group-item"><a th:href="@{/servicos}">Serviços</a></li>  
 <li class="list-group-item"><a th:href="@{/agendamentos}">Agendamentos</a></li>  
 <li class="list-group-item"><a th:href="@{/produtos}">Produtos</a></li>  
 <li class="list-group-item"><a th:href="@{/vendas}">Vendas</a></li>  
 </ul>  
 </div>  
</body>  
</html>**

**7.2. Lista e Formulário de Agendamentos**

**lista.html (em templates/agendamentos/lista.html):**

**<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Lista de Agendamentos</title>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">  
</head>  
<body>  
 <div class="container mt-4">  
 <h2>Agendamentos</h2>  
 <a th:href="@{/agendamentos/novo}" class="btn btn-primary mb-3">Novo Agendamento</a>  
 <table class="table table-bordered">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Serviço</th>  
 <th>Cliente</th>  
 <th>Data/Hora</th>  
 <th>Ações</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="agendamento : ${agendamentos}">  
 <td th:text="${agendamento.id}"></td>  
 <td th:text="${agendamento.servico.nome}"></td>  
 <td th:text="${agendamento.clienteNome}"></td>  
 <td th:text="${agendamento.dataHora}"></td>  
 <td>  
 <a th:href="@{'/agendamentos/editar/' + ${agendamento.id}}" class="btn btn-sm btn-warning">Editar</a>  
 <a th:href="@{'/agendamentos/deletar/' + ${agendamento.id}}" class="btn btn-sm btn-danger">Deletar</a>  
 </td>  
 </tr>  
 </tbody>  
 </table>  
 <a th:href="@{/}" class="btn btn-secondary">Voltar</a>  
 </div>  
</body>  
</html>**

**form.html (em templates/agendamentos/form.html):**

**<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Formulário de Agendamento</title>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">  
</head>  
<body>  
 <div class="container mt-4">  
 <h2 th:text="${agendamento.id == null} ? 'Novo Agendamento' : 'Editar Agendamento'"></h2>  
 <form th:action="@{/agendamentos/salvar}" th:object="${agendamento}" method="post">  
 <div class="mb-3">  
 <label class="form-label">Serviço</label>  
 <select class="form-select" th:field="\*{servico}" required>  
 <option th:each="servico : ${servicos}" th:value="${servico}" th:text="${servico.nome}"></option>  
 </select>  
 </div>  
 <div class="mb-3">  
 <label class="form-label">Nome do Cliente</label>  
 <input type="text" class="form-control" th:field="\*{clienteNome}" required>  
 </div>  
 <div class="mb-3">  
 <label class="form-label">Data e Hora</label>  
 <input type="datetime-local" class="form-control" name="dataHoraStr" required>  
 </div>  
 <button type="submit" class="btn btn-success">Salvar</button>  
 <a th:href="@{/agendamentos}" class="btn btn-secondary">Cancelar</a>  
 </form>  
 </div>  
</body>  
</html>**

**Observação: Crie páginas similares para os demais módulos (clientes, funcionários, produtos, vendas e serviços). Mantenha o design simples usando Bootstrap e a lógica básica de formulários e listagens.**

**8. Execução da Aplicação**

**8.1. Classe Principal**

**Crie a classe principal (por exemplo, em com.seuprojeto.esteticaauto.EsteticaAutoApplication.java):**

**package com.seuprojeto.esteticaauto;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class EsteticaAutoApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(EsteticaAutoApplication.class, args);  
 }  
}**

**8.2. Executando o Projeto**

**No terminal (na raiz do projeto), execute:**

**mvn spring-boot:run**

**O Spring Boot iniciará um servidor embutido (geralmente Tomcat) na porta 8080. Agora você poderá acessar a aplicação via:**

**http://localhost:8080**

**9. Acesso Remoto e Deploy**

Para acessar de outros dispositivos na mesma rede:

Certifique-se de que o firewall da sua máquina permita conexões na porta 8080.

Descubra o IP local da sua máquina (por exemplo, 192.168.1.100) e acesse a aplicação via:

**http://192.168.1.100:8080**

Caso queira disponibilizar na internet, será necessário configurar um domínio e encaminhar as portas, ou fazer o deploy em um servidor/cloud (como Heroku, AWS, etc.).

**10. Conclusão e Próximos Passos**

Com esse passo a passo, você terá:

Um projeto Spring Boot configurado com Java 21.

Conexão com o banco MySQL.

Entidades mapeadas para cada tabela do banco.

Repositórios para facilitar as operações CRUD.

Controllers e páginas Thymeleaf para exibir, criar, editar e deletar registros.

Uma aplicação acessível via navegador a partir de um link.

Esse escopo básico pode ser expandido futuramente (por exemplo, melhorando a validação, segurança, ou adicionando funcionalidades extras).