


Sistema de Agendamento para Estética Automotiva

Este projeto apresenta um sistema completo para gerenciamento de serviços de estética automotiva, desenvolvido como projeto final do curso Técnico em Desenvolvimento de Sistemas do Senai. A aplicação utiliza Java 21, Spring Boot, Thymeleaf e MySQL para criar uma solução robusta de agendamentos.

O sistema inclui módulos para gerenciamento de clientes, agendamentos, produtos, vendas, serviços e funcionários, oferecendo uma solução completa para empresas do setor de estética automotiva.

 **por Lucas Matheus Rodrigues de Jesus**



Pré-requisitos e Criação do Projeto



Java 21

Certifique-se de ter o JDK 21 instalado em sua máquina para compatibilidade com o projeto.



MySQL Workbench

Banco de dados relacional para armazenamento das informações do sistema.



VS Code

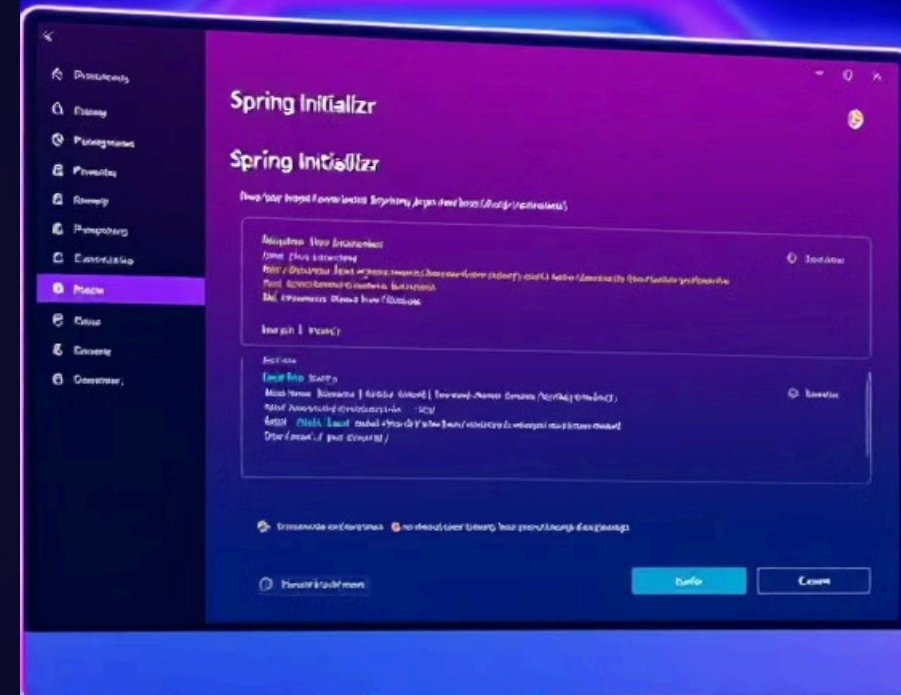
IDE recomendada para desenvolvimento e execução do projeto.



Maven

Utilizado para gerenciamento de dependências e build do projeto.

O projeto é criado utilizando o Spring Initializr com configurações específicas: Maven Project, Java como linguagem, Spring Boot na versão mais recente compatível com Java 21, e dependências essenciais como Spring Web, Spring Data JPA, Thymeleaf e MySQL Driver.





Configuração do Banco de Dados

Conexão MySQL

A configuração é realizada através do arquivo `application.properties`, onde definimos a URL de conexão com o MySQL, credenciais de acesso e comportamento do JPA.

Configuração JPA

Como estamos trabalhando com um banco já existente, configuramos o hibernate para não modificar o esquema (`ddl-auto=none`).

Credenciais

É importante substituir `SEU_USUARIO` e `SUA_SENHA` pelas credenciais reais do seu MySQL para garantir o acesso correto ao banco de dados.

A configuração adequada do banco de dados é essencial para o funcionamento correto do sistema, garantindo que todas as operações de persistência sejam realizadas de forma eficiente e segura.

```
estetica > src > main > resources > application.properties
1  spring.application.name=EsteticaAutomotiva
2  spring.datasource.url=jdbc:mysql://127.0.0.1:3306/estetica_auto?user=luкас
3  spring.datasource.username=root
4  spring.datasource.password=
5  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6
7  # Como o banco já existe e está populado, não queremos que o JPA modifique o esquema:
8  spring.jpa.hibernate.ddl-auto=none
9
10 # Exibir as queries no console (opcional)
11 spring.jpa.show-sql=true
12 spring.jpa.properties.hibernate.format_sql=true
13
```

Criação das Entidades (Model)

1

Cliente

Armazena informações dos clientes como nome, telefone, email e veículo. Relaciona-se com agendamentos e vendas.

2

Produto

Representa os produtos disponíveis para venda, contendo id, nome e preço. Os produtos são utilizados nas vendas realizadas para os clientes.

3

Serviço

Contém os serviços oferecidos pela estética automotiva, com nome, descrição e preço. Relaciona-se com agendamentos.

4

Agendamento

Representa os agendamentos de serviços, relacionando clientes e serviços com data e hora específicas.

Cada entidade é mapeada para uma tabela correspondente no banco de dados usando anotações JPA como @Entity, @Table, @Id e @GeneratedValue. Os relacionamentos entre entidades são estabelecidos usando @ManyToOne e @JoinColumn.

Repositórios e Controllers

Repositórios

Para cada entidade, criamos uma interface que estende JpaRepository, fornecendo métodos CRUD prontos para uso:

- ClienteRepository
- ProdutoRepository
- ServicoRepository
- AgendamentoRepository
- VendaRepository
- FuncionarioRepository

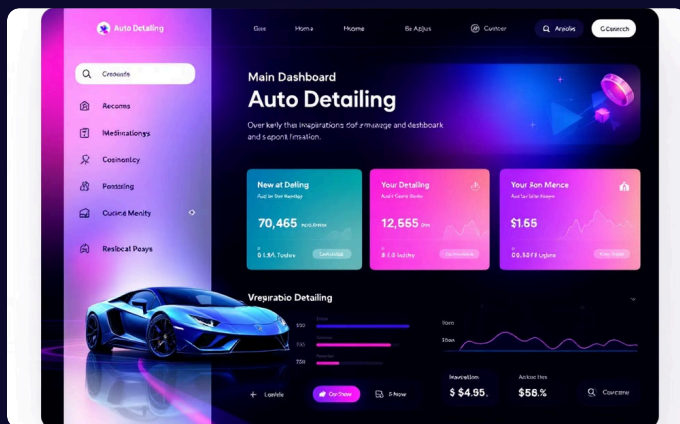
Controllers

Os controllers gerenciam as requisições HTTP e a lógica de negócio:

- Listar todos os registros
- Exibir formulários para criação
- Salvar novos registros
- Editar registros existentes
- Deletar registros

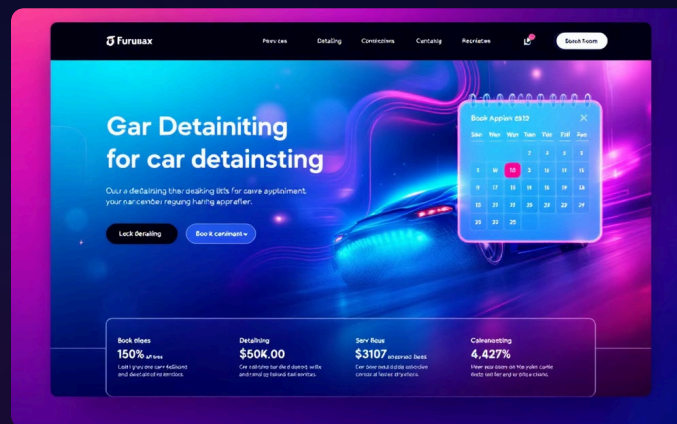
Cada controller utiliza o repositório correspondente para acessar os dados e o Thymeleaf para renderizar as páginas HTML. Os controllers são anotados com @Controller e @RequestMapping para definir as rotas.

Páginas Thymeleaf



[Página Principal](#)

A página inicial (index.html) apresenta links para todos os módulos do sistema, permitindo navegação fácil entre as diferentes funcionalidades.



Formulários

Formulários para criação e edição de registros, com campos específicos para cada entidade e validações básicas para garantir a integridade dos dados.



Páginas de Listagem

Cada módulo possui uma página de listagem que exibe todos os registros em formato de tabela, com opções para adicionar, editar e excluir.

As páginas Thymeleaf combinam HTML com expressões especiais que permitem a integração com o backend Java. O Bootstrap é utilizado para estilização, garantindo uma interface responsiva e moderna.

Problemas a resolver

No teste feito no sistema, notamos que as classes não deletava, pois as entidades estavam relacionadas e não permitia excluir sem desvincular na entidade relacionada.

Solução: Usar DELETE com JavaScript (Recomendado)

Para seguir as boas práticas RESTful, use o método HTTP DELETE:

1. Template (Thymeleaf):

```
<form th:action="@{/clientes/' + ${cliente.id}'}" method="post">
  <input type="hidden" name="_method" value="delete"/> <!-- Ativa o método DELETE -->
  <button type="submit" class="btn btn-sm btn-outline-danger">
    <i class="fas fa-trash-alt"></i>
  </button>
</form>
```

1. No Controller:

```
@DeleteMapping("/{id}") // Usando DELETE
public String deletarCliente(@PathVariable Long id) {
    // ...
}
```

1. Habilitar Suporte a Métodos HTTP no application.properties:

```
spring.mvc.hiddenmethod.filter.enabled=true
```

Passos Adicionais (Se Usar Spring Security):

Se estiver usando Spring Security, adicione o token CSRF ao formulário:

```
<form th:action="@{'/clientes/deletar/' + ${cliente.id}}" method="post">
  <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
  <button type="submit" class="btn btn-sm btn-outline-danger">
    <i class="fas fa-trash-alt"></i>
  </button>
</form>
```


Exemplo Final Funcional (POST):

Controller:

```
@Controller
@RequestMapping("/clientes")
public class ClienteController {

    @Autowired
    private ClienteRepository clienteRepository;

    @PostMapping("/deletar/{id}")
    public String deletarCliente(@PathVariable Long id) {
        clienteRepository.deleteById(id);
        return "redirect:/clientes";
    }
}
```

Template:

```
<form th:action="@{'/clientes/deletar/' + ${cliente.id}}" method="post">
    <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
    <button type="submit" class="btn btn-sm btn-danger">Deletar</button>
</form>
```

Execução e Acesso ao Sistema

Classe Principal

A classe `EsteticaAutoApplication` contém o método `main` que inicia a aplicação Spring Boot, configurando automaticamente o servidor embutido e as dependências.

Executando o Projeto

No terminal, na raiz do projeto, execute o comando `mvn spring-boot:run` para iniciar o servidor na porta 8080.

Acesso Local

Acesse a aplicação através do navegador pelo endereço `http://localhost:8080` para utilizar todas as funcionalidades do sistema.

Acesso Remoto

Para acessar de outros dispositivos na mesma rede, utilize o IP da máquina (ex: `http://192.168.1.100:8080`), certificando-se que o firewall permite conexões na porta 8080.

O Spring Boot inicia um servidor Tomcat embutido, tornando a aplicação acessível via navegador. Para disponibilizar na internet, seria necessário configurar um domínio e encaminhar portas, ou fazer deploy em um servidor/cloud como Heroku ou AWS.

Conclusão e Próximos Passos

Sistema Completo

Projeto Spring Boot configurado com Java 21, conectado ao MySQL, com entidades mapeadas e interface web funcional.

Expansão Futura

Possibilidades de melhorias em validação, segurança e adição de novas funcionalidades.



Funcionalidades CRUD

Operações completas de Criar, Ler, Atualizar e Deletar para todos os módulos do sistema.

Interface Amigável

Páginas Thymeleaf com Bootstrap proporcionando uma experiência de usuário moderna e responsiva.

Este sistema de agendamento para estética automotiva oferece uma solução completa para gerenciamento de clientes, serviços, produtos, vendas e funcionários. Como próximos passos, poderia-se implementar autenticação de usuários, relatórios gerenciais, integração com sistemas de pagamento e uma versão mobile para acesso em dispositivos móveis.