Serviço Nacional de Aprendizagem Industrial

Elessandro de Abreu
Gustavo de Oliveira Neves
Jean Matei Rodrigues
Samara Bento
Tainan Almeida

RELATÓRIO SISTÊMICO

SMR-AUTO

TUBARÃO

2025

Elessandro de Abreu
Gustavo de Oliveira Neves
Jean Matei Rodrigues
Samara Bento
Tainan Almeida

RELATÓRIO SISTÊMICO

SMR-AUTO

Relatório apresentado à disciplina Desenvolvimento de Sistemas, do curso de Técnico de Analise e Desenvolvimento de Sistemas do SENAI como primeira atividade prática, sob a orientação do prof.º Matheus da S. Carvalho.

TUBARÃO

2025

RESUMO

Este relatório apresenta a documentação técnica de um sistema de gerenciamento desenvolvido para oficinas mecânicas, denominado SMR AUTO. O sistema foi projetado para otimizar processos administrativos e operacionais comuns em ambientes de mecânica automotiva, incluindo gerenciamento de clientes, funcionários, peças e ordens de serviço. Utilizando uma arquitetura cliente-servidor com Spring Boot no back-end e JavaScript/HTML/CSS no front-end, o sistema implementa funcionalidades essenciais como cadastro de clientes, funcionários e peças, emissão de ordens de serviço com cálculos automáticos, controle de estoque e relatórios básicos. A documentação abrange análise detalhada da arquitetura, estrutura de código, modelos de dados, interfaces de usuário e considerações de usabilidade. Os resultados demonstram que o sistema é capaz de atender às necessidades fundamentais de uma oficina mecânica de pequeno a médio porte, proporcionando ganhos significativos em eficiência operacional e organização de informações.

Palavras-chave: Sistema de Gerenciamento, Oficina Mecânica, Spring Boot, JavaScript, Controle de Estoque, Ordens de Serviço.

LISTA DE ILUSTRAÇÕES

Sumário

1. INTRODUÇÃO

1.1 Contextualização

O setor de oficinas mecânicas automotivas no Brasil caracteriza-se por uma diversidade de estabelecimentos, desde pequenas oficinas familiares até grandes centros automotivos. Independentemente do tamanho, a gestão eficiente de informações, recursos e processos representa um desafio significativo para estes negócios. Muitas oficinas ainda dependem de métodos manuais ou sistemas improvisados para gerenciar seus clientes, estoque, serviços e faturamento.

Neste contexto, o sistema SMR AUTO foi desenvolvido como solução específica para este segmento, visando informatizar e otimizar os principais processos operacionais e administrativos de uma oficina mecânica. O sistema busca proporcionar uma ferramenta acessível e prática que atenda às necessidades específicas deste tipo de negócio.

1.2 Objetivos

O objetivo geral deste trabalho é documentar e analisar o desenvolvimento de um sistema de gerenciamento para oficinas mecânicas que permita o controle eficiente de clientes, funcionários, peças e ordens de serviço.

Os objetivos específicos incluem:

- Descrever a arquitetura e os componentes do sistema;
- Apresentar as funcionalidades implementadas;
- Analisar o modelo de dados utilizado;
- Detalhar os aspectos técnicos da implementação frontend e backend;
- Avaliar os resultados obtidos em termos de desempenho e usabilidade;
- Identificar limitações e possibilidades de melhorias futuras.

1.3 Justificativa

A implementação de um sistema de gestão específico para oficinas mecânicas justifica-se pela necessidade de:

- Organizar informações de clientes, veículos e histórico de serviços;
- Controlar eficientemente o estoque de peças e componentes;
- Agilizar a criação e acompanhamento de ordens de serviço;
- Facilitar o cálculo de valores de serviços e aplicação de descontos;

- Manter registros precisos de funcionários e suas atribuições;
- Permitir consultas rápidas a dados históricos;
- Proporcionar uma interface simples e intuitiva, adequada ao perfil dos usuários típicos deste segmento.

O sistema busca suprir estas necessidades de forma integrada, proporcionando uma solução tecnológica acessível para a modernização da gestão de oficinas mecânicas.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas de Gestão para Oficinas Mecânicas

Os sistemas de gestão para oficinas mecânicas evoluíram significativamente nas últimas décadas, acompanhando tanto os avanços tecnológicos quanto as crescentes complexidades do setor automotivo. Estes sistemas são ferramentas especializadas que auxiliam na organização e controle de todas as operações típicas de uma oficina, desde o agendamento de serviços até o controle financeiro.

De acordo com Silva (2019), um sistema eficiente para oficinas mecânicas deve contemplar, no mínimo, os seguintes módulos: cadastro de clientes e veículos, controle de estoque, gestão de serviços, controle financeiro e relatórios gerenciais. Todos estes aspectos foram considerados no desenvolvimento do SMR AUTO, com foco especial na usabilidade e simplicidade de operação.

2.2 Arquitetura Cliente-Servidor

O sistema SMR AUTO foi desenvolvido utilizando a arquitetura cliente-servidor, modelo que separa a aplicação em duas partes principais: o cliente (frontend) e o servidor (backend). Segundo Tanenbaum e Van Steen (2007), esta arquitetura apresenta vantagens significativas para sistemas empresariais, incluindo centralização do controle, escalabilidade e facilidade de manutenção.

No contexto do SMR AUTO, o servidor (backend) foi implementado em Java utilizando o framework Spring Boot, responsável pelo processamento das regras de negócio, acesso a dados e exposição de APIs REST. O cliente (frontend) foi desenvolvido utilizando JavaScript, HTML e CSS, sendo responsável pela interface com o usuário e comunicação com o servidor via requisições HTTP.

2.3 Tecnologias Utilizadas

2.3.1 Backend

O backend do sistema foi desenvolvido utilizando as seguintes tecnologias:

- **Java**: Linguagem de programação orientada a objetos, escolhida por sua robustez, portabilidade e amplo suporte a aplicações empresariais.
- **Spring Boot**: Framework que simplifica o desenvolvimento de aplicações Java, oferecendo configuração automática e um ecossistema rico de ferramentas.
- **Spring Data JPA**: Facilita a implementação de repositórios de acesso a dados, reduzindo significativamente o código boilerplate necessário.
- **Hibernate**: Framework ORM (Object-Relational Mapping) utilizado para mapeamento objeto-relacional.
- **MySQL**: Sistema de gerenciamento de banco de dados relacional utilizado para persistência dos dados.

2.3.2 Frontend

O frontend do sistema foi desenvolvido utilizando:

- **HTML5**: Linguagem de marcação para estruturação do conteúdo das páginas.
- **CSS3**: Linguagem de estilo utilizada para definir a apresentação visual do sistema.
- **JavaScript**: Linguagem de programação utilizada para implementar a interatividade e comportamento dinâmico do sistema.
- **Fetch API**: Interface JavaScript para realizar requisições HTTP, utilizada para comunicação com o backend.
- **Font Awesome**: Biblioteca de ícones vetoriais utilizada na interface gráfica.

3. DESCRIÇÃO DO SISTEMA

3.1 Visão Geral

O SMR AUTO é um sistema de gerenciamento desenvolvido especificamente para oficinas mecânicas, com o objetivo de informatizar e otimizar os principais processos operacionais e administrativos deste tipo de negócio. O sistema permite o cadastro e gerenciamento de clientes, funcionários, peças e ordens de serviço, oferecendo uma solução completa para as necessidades básicas de uma oficina mecânica.

A aplicação possui interface web responsiva, permitindo acesso a partir de diferentes dispositivos, e utiliza uma arquitetura cliente-servidor para processamento e persistência de dados. O sistema foi projetado priorizando a simplicidade e facilidade de uso, considerando o perfil dos usuários típicos de oficinas mecânicas.

3.2 Arquitetura

A arquitetura do sistema segue o padrão MVC (Model-View-Controller), com separação clara entre as camadas de apresentação, lógica de negócio e acesso a dados.

O sistema é composto pelos seguintes componentes principais:

- **Frontend (Camada de Apresentação)**: Implementado com HTML, CSS e JavaScript, responsável pela interface com o usuário.
- **Backend (Camada de Lógica de Negócio)**: Implementado com Spring Boot, responsável pelo processamento das regras de negócio e coordenação do sistema.
- **Repositórios (Camada de Acesso a Dados)**: Implementados com Spring Data JPA, responsáveis pela persistência e recuperação de dados.
- **Banco de Dados**: Utiliza MySQL para armazenamento persistente dos dados do sistema.

A comunicação entre o frontend e o backend é realizada através de uma API REST, que expõe endpoints para as diferentes funcionalidades do sistema.

3.3 Funcionalidades

O sistema SMR AUTO implementa as seguintes funcionalidades principais:

3.3.1 Gestão de Clientes

- Cadastro de dados pessoais e de contato

- Registro de endereço completo
- Busca por nome ou CPF

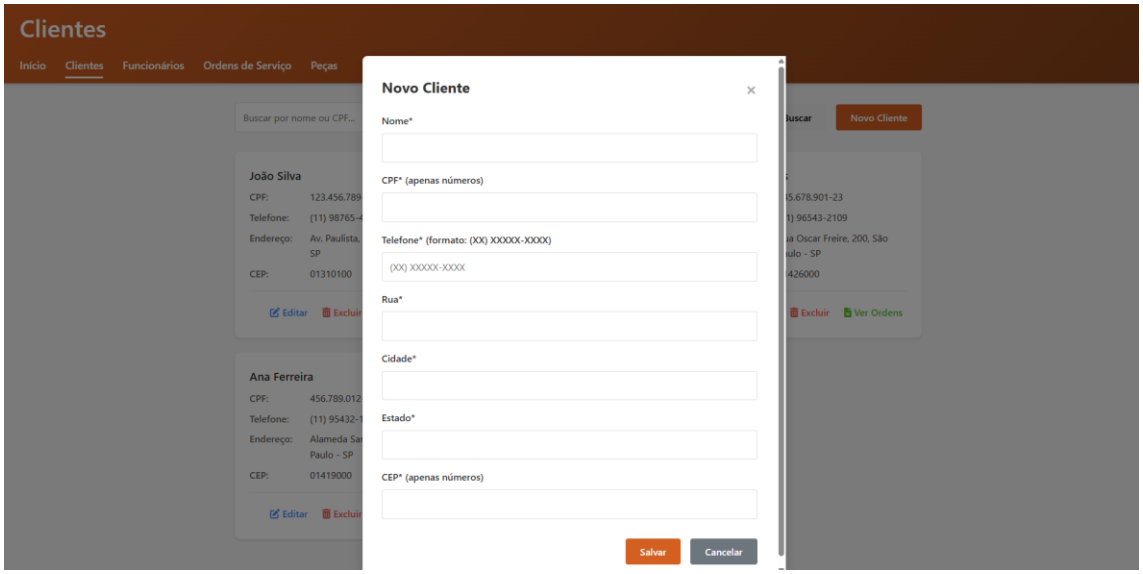


FIGURA 1 - TELA DE CADASTRO

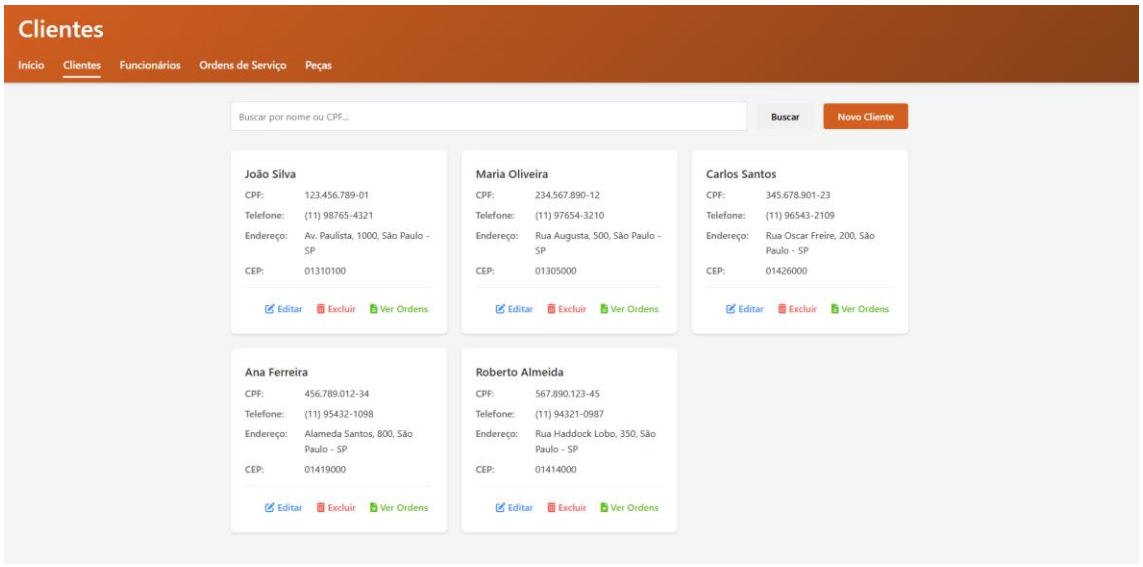


FIGURA 2 - TELA DE CONSULTA DE CLIENTES

- Edição e exclusão de registros. Na tela de exclusão, apesar de ser a mesma da listagem aparece na parte superior um pedido de confirmação de exclusão.

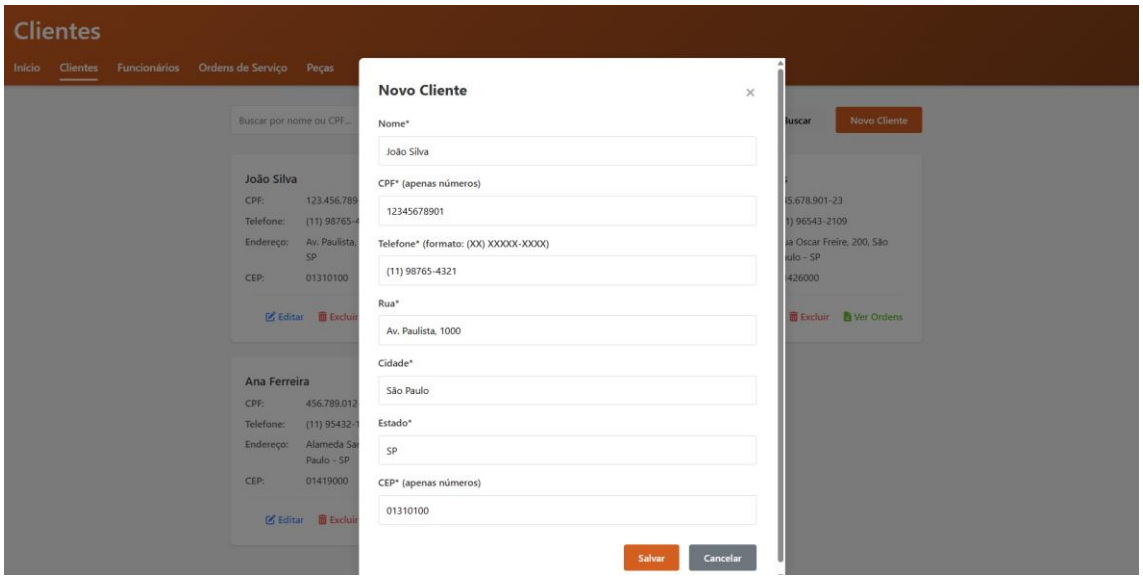


FIGURA 3 - TELA DE EDIÇÃO DE CLIENTE

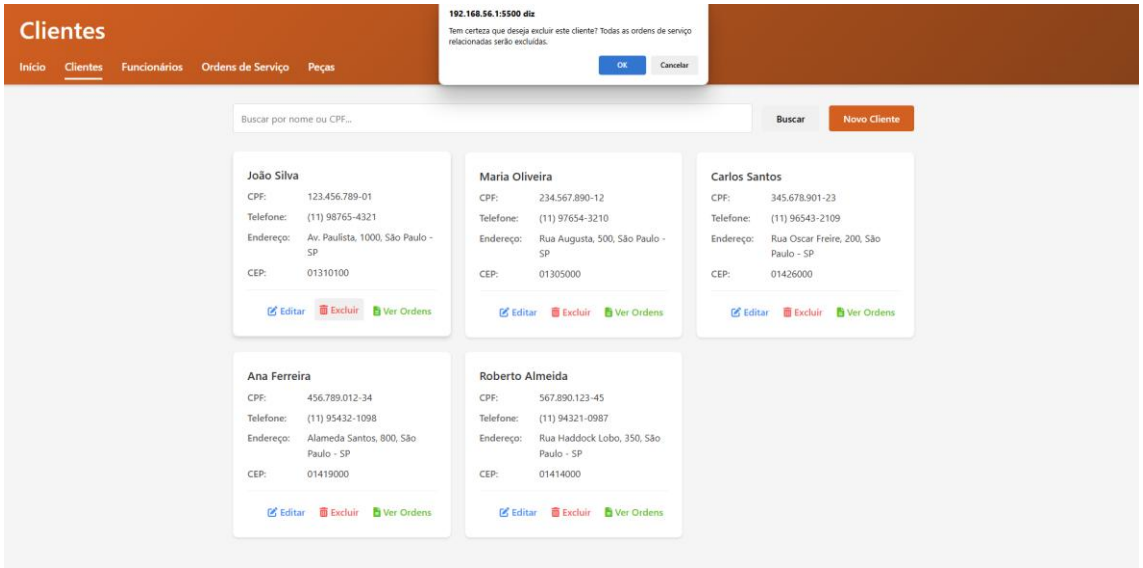


FIGURA 4 - TELA DE EXCLUSÃO DE CLIENTE

- Visualização de ordens de serviço associadas

3.3.2 Gestão de Funcionários

- Cadastro de dados pessoais e de contato
- Registro de cargo/função
- Busca por nome, cargo ou CPF

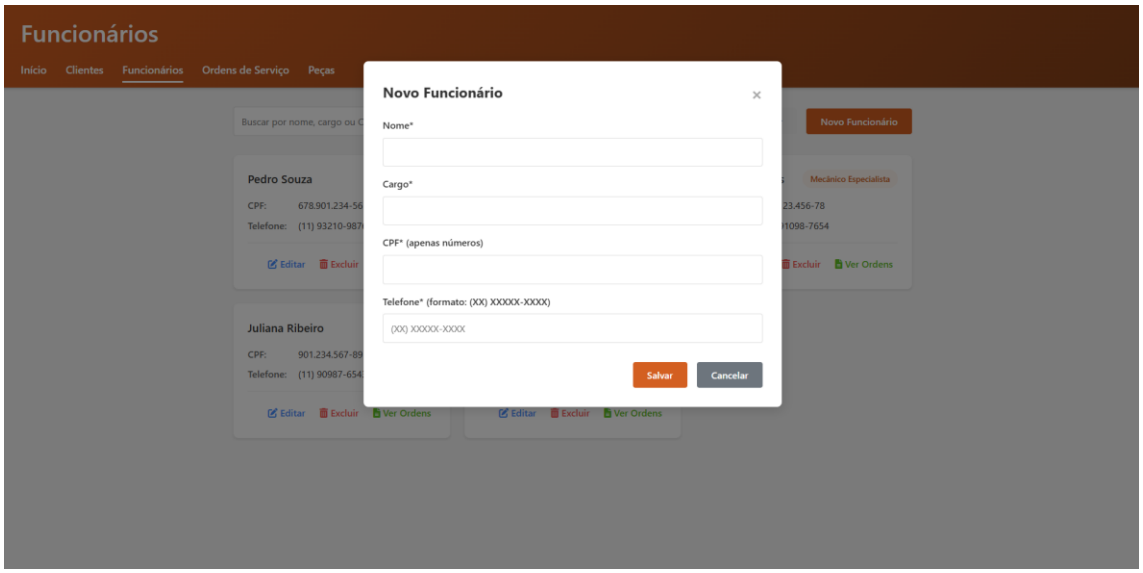


FIGURA 5 - TELA CADASTRO DE FUNCIONÁRIO

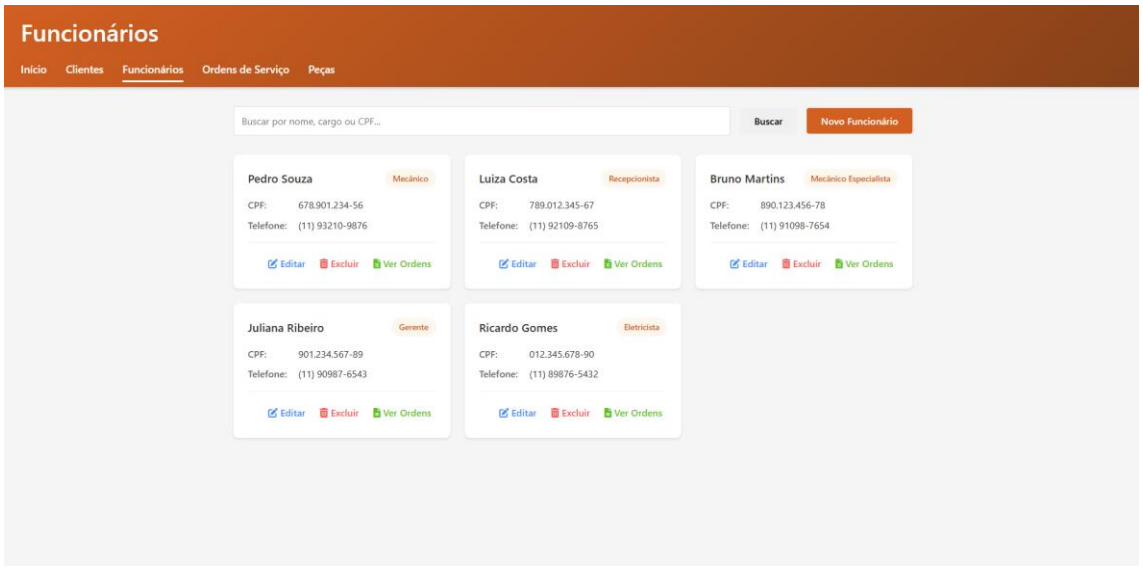


FIGURA 6 - TELA DE LISTAGEM DE FUNCIONÁRIO

- Edição e exclusão de registros. Na tela de exclusão, apesar de ser a mesma da listagem aparece na parte superior um pedido de confirmação de exclusão.

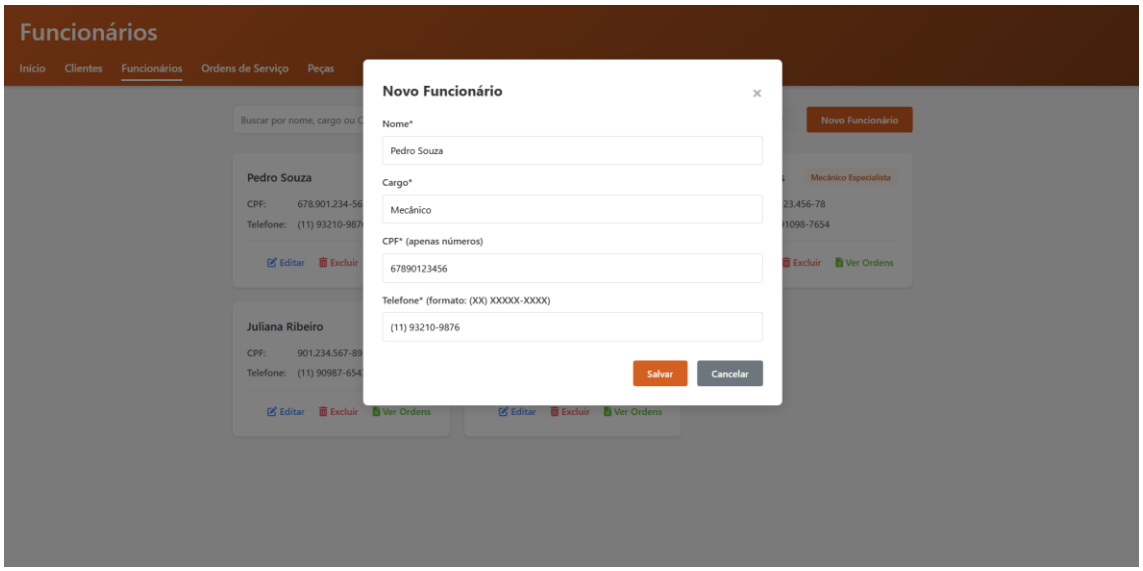


FIGURA 7 - TELA DE EDIÇÃO DE FUNCIONÁRIO

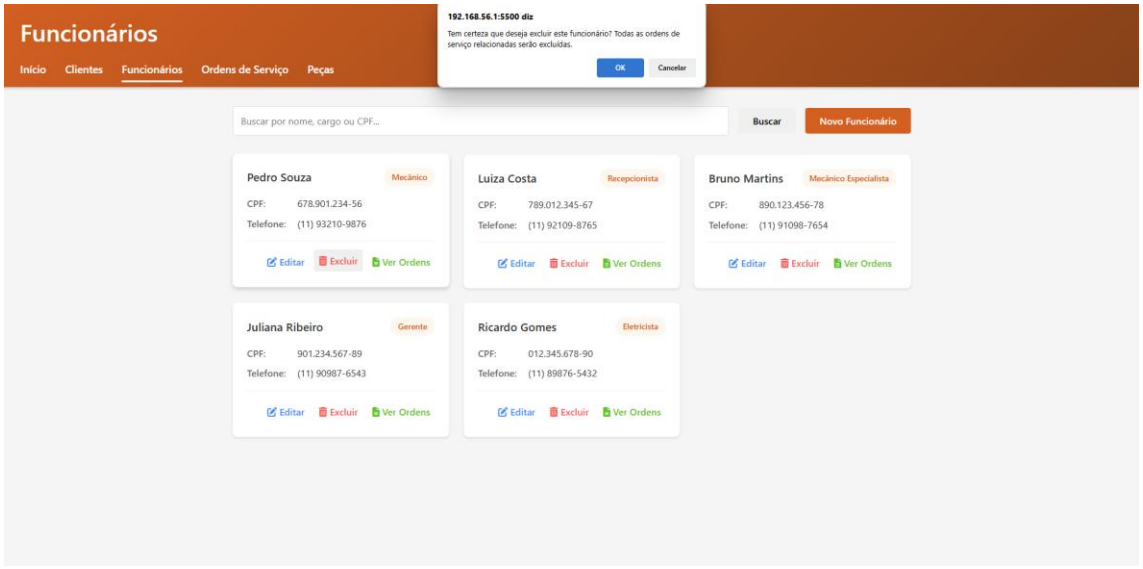


FIGURA 8 - TELA DE EXCLUSÃO DE FUNCIONÁRIO

3.3.3 Gestão de Peças

- Cadastro de peças com descrição e valores

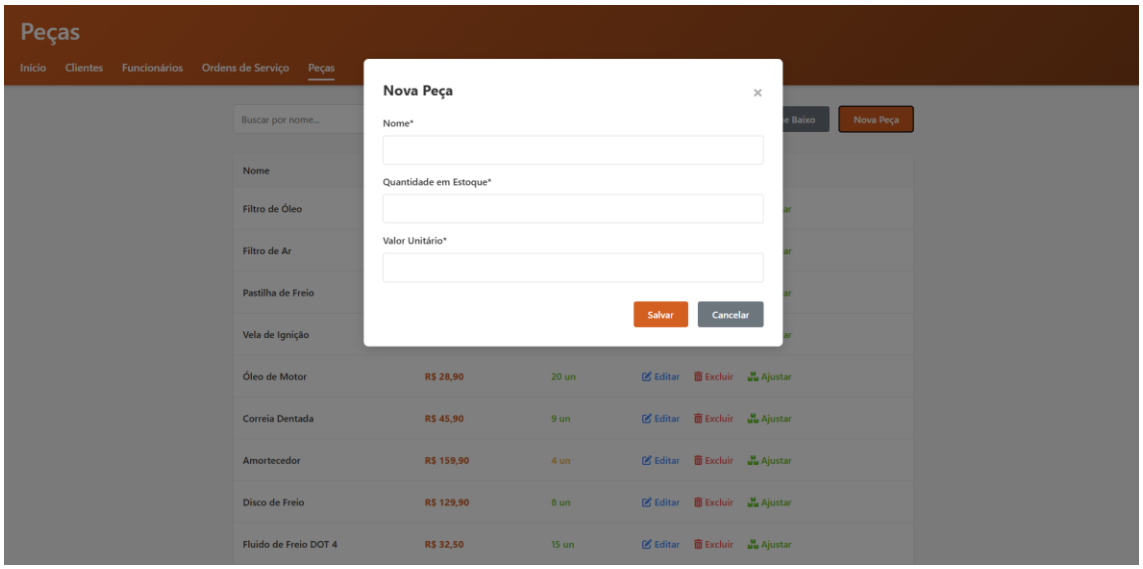


FIGURA 9 - TELA DE CADASTRO DE PEÇAS

- Controle de estoque

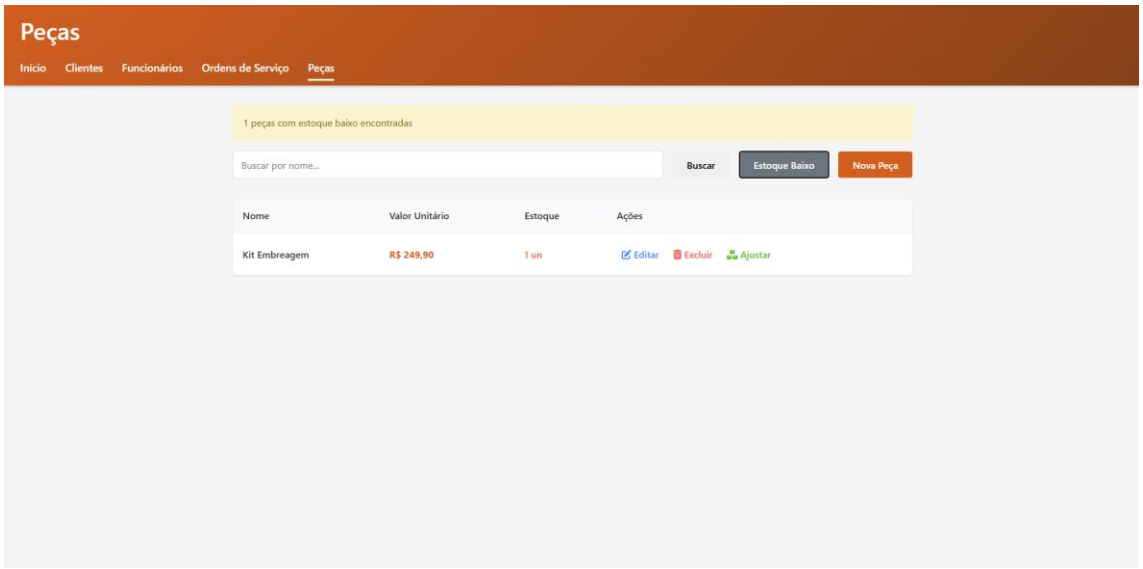


FIGURA 10 - TELA DE CONTROLE DE ESTOQUE

- Alerta de estoque baixo, nessa tela o alerta fica visível com o padrão de cores, sendo a cor vermelha destinada para o estoque baixo ou zona de risco, a cor amarela para indicar um estoque perto da zona de risco e verde para indicar um estoque adequado.

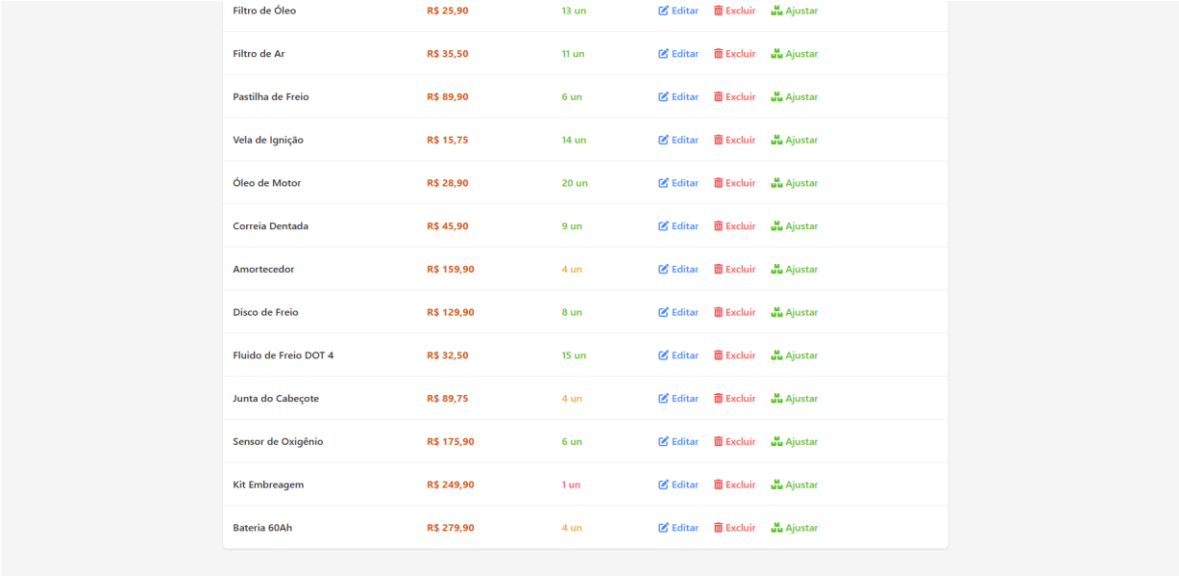


FIGURA 11 - TELA DE ALERTA DE ESTOQUE

- Ajuste manual de quantidades, esta função é apenas uma etapa para facilitar e não precisar abrir a aba inteira da peça.

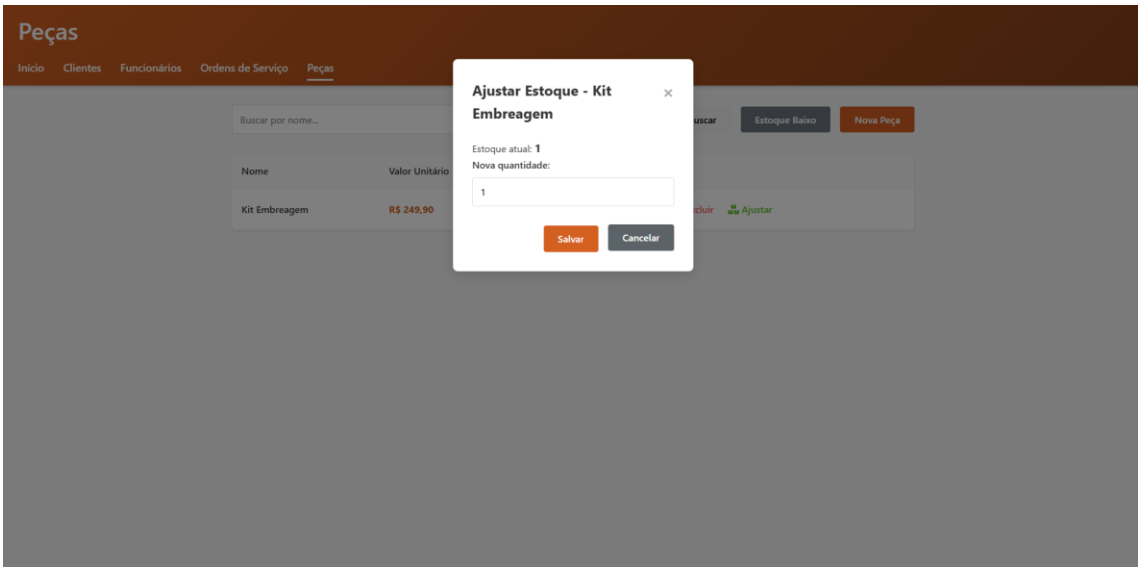


FIGURA 12 - TELA DE AJUSTE MANUAL DE ESTOQUE

- Busca por nome

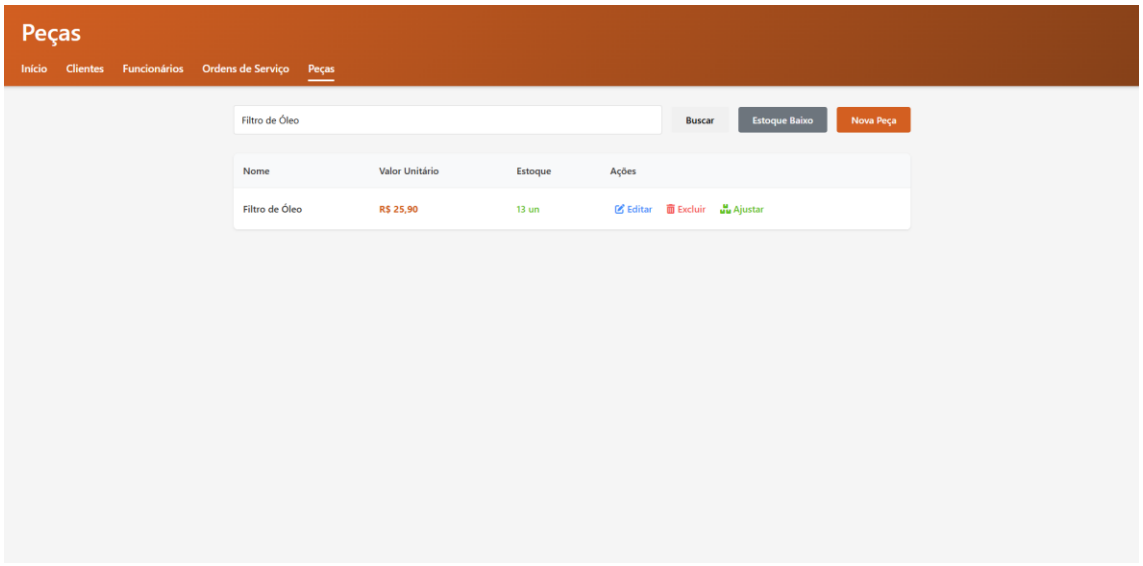


FIGURA 13 - TELA DE BUSCA

- Edição e exclusão de registros. Na tela de exclusão, apesar de ser a mesma da listagem aparece na parte superior um pedido de confirmação de exclusão.

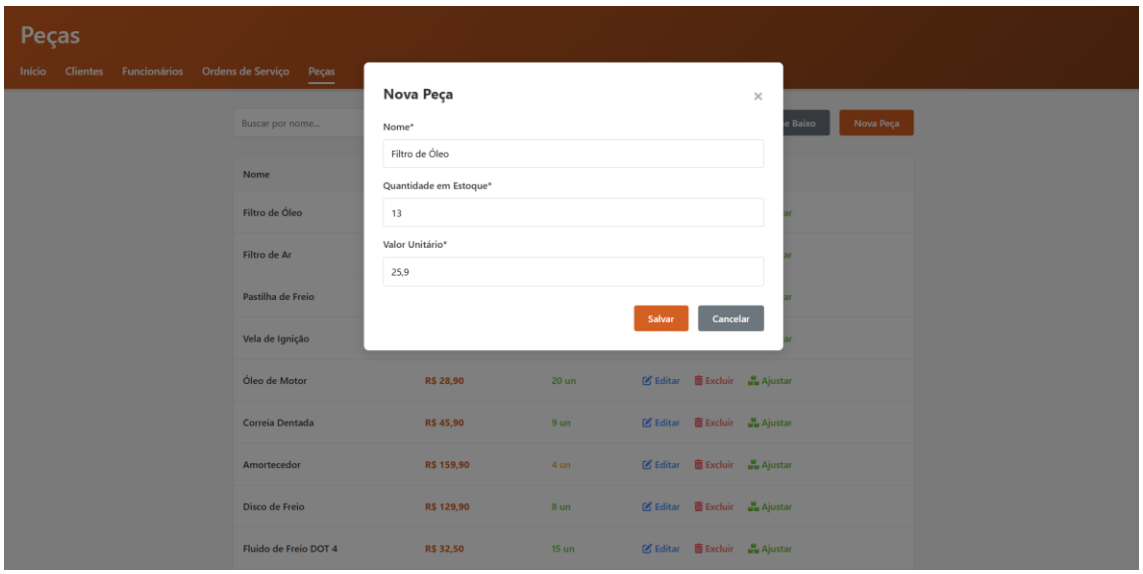


FIGURA 14 - TELA DE ALTERAÇÃO DE PEÇA

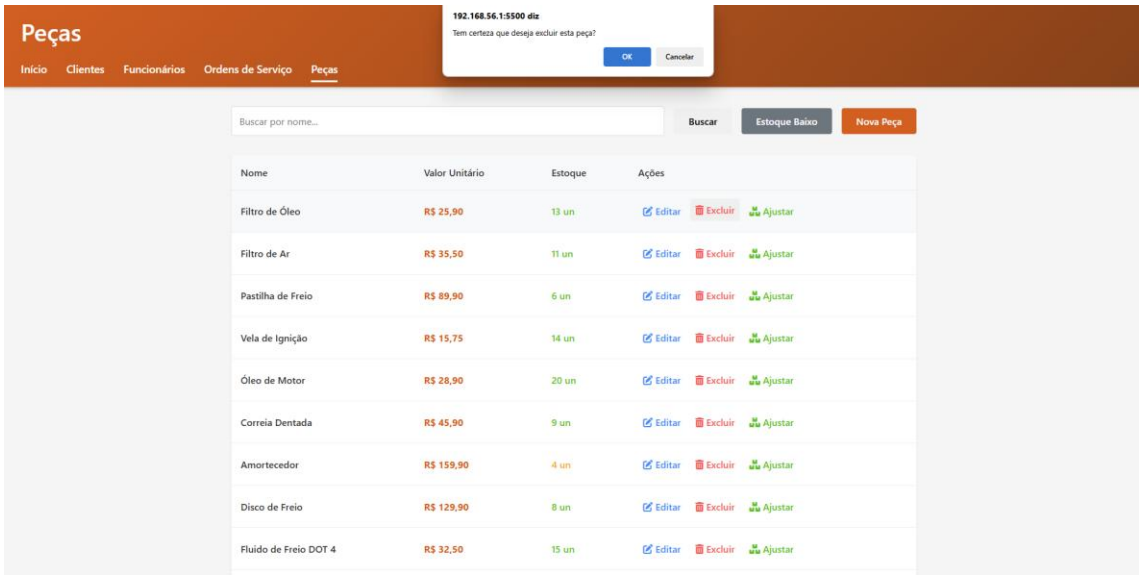


FIGURA 15 - TELA DE EXCLUSÃO DE PEÇA

3.3.4 Gestão de Ordens de Serviço

- Criação de ordens vinculadas a clientes e funcionários
- Registro de veículo e descrição do problema
- Inclusão de peças utilizadas com controle automático de estoque
- Registro de valor de mão de obra
- Cálculo automático de valores e descontos

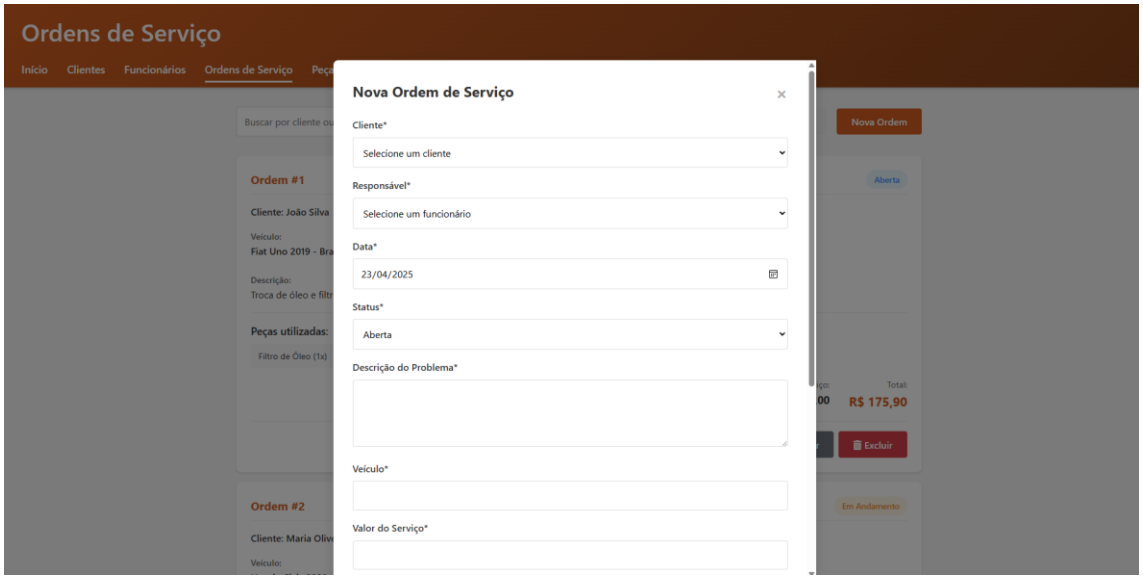


FIGURA 16 - TELA DE CADASTRO DE ORDEM DE SERVIÇO

- Busca de ordens por cliente

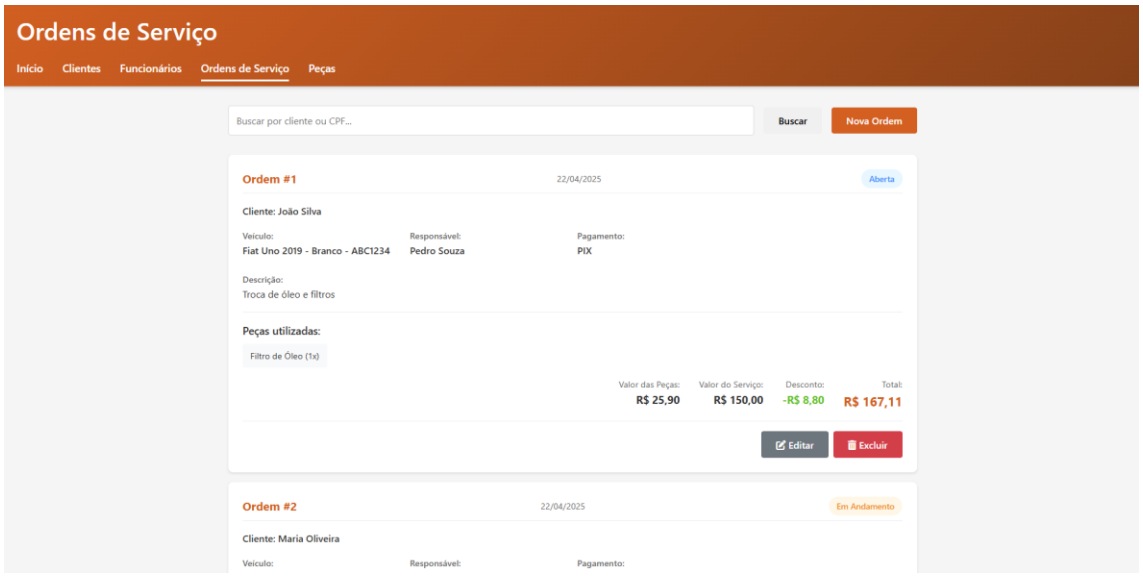


FIGURA 17 - TELA DE LISTAGEM DE CLIENTES

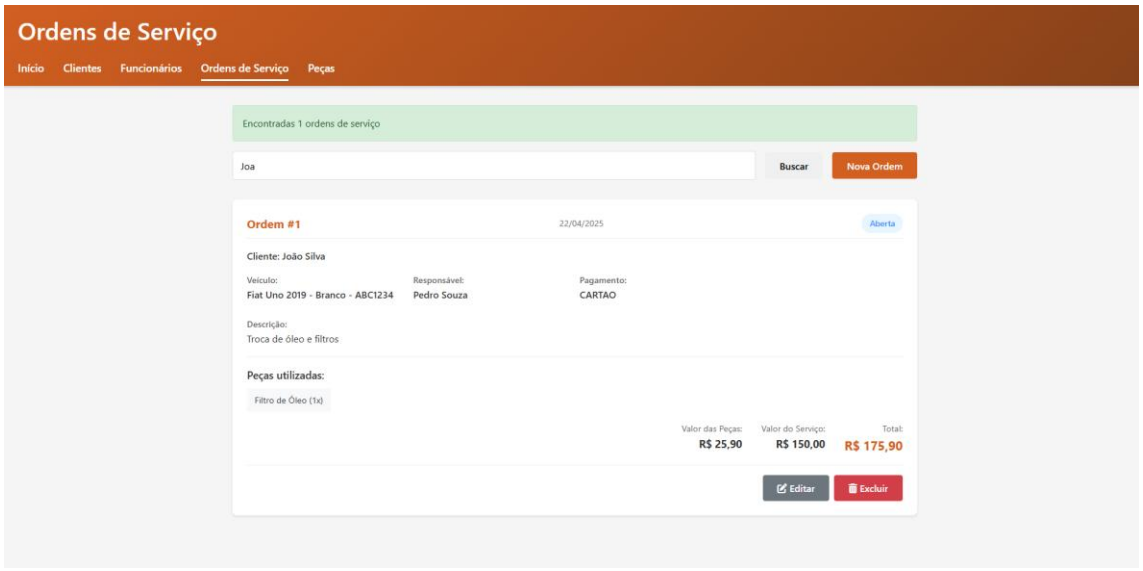


FIGURA 18 - TELA DE BUSCA DE ORDEM DE SERVIÇO

- Edição e exclusão de ordens de serviço. Na tela de exclusão, apesar de ser a mesma da listagem aparece na parte superior um pedido de confirmação de exclusão.

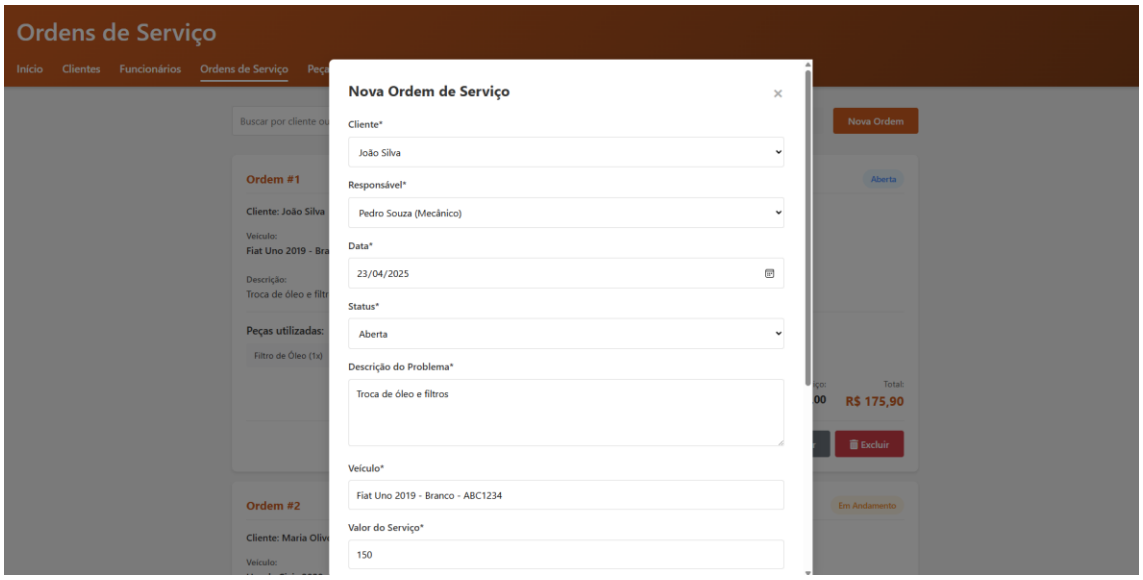


FIGURA 19 - TELA DE ALTERAÇÃO DE ORDEM DE SERVIÇO

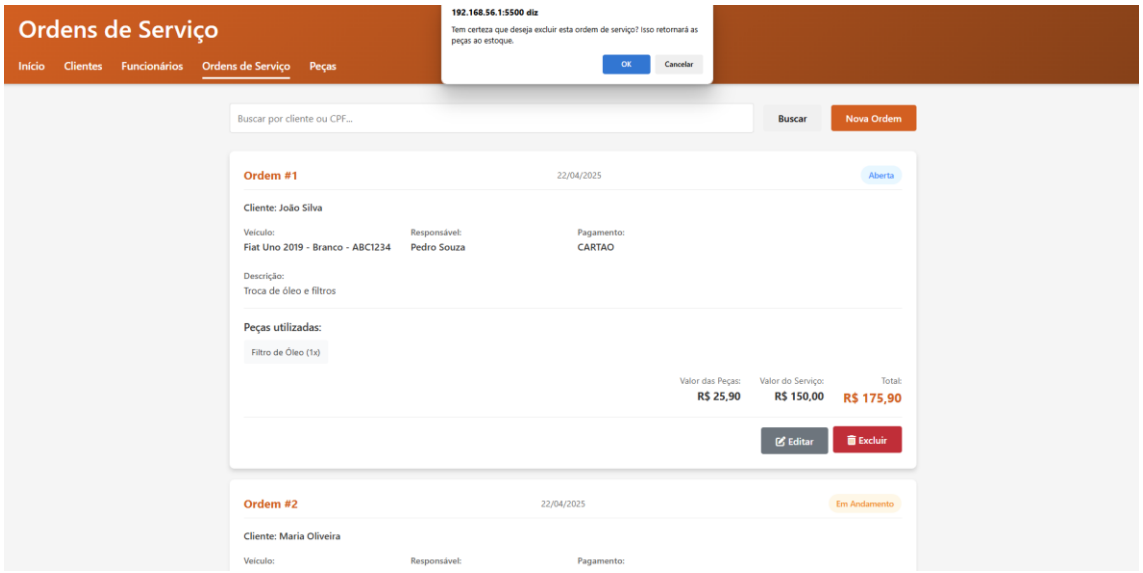


FIGURA 20 - TELA DE EXCLUSÃO DE ORDEM DE SERVIÇO

3.4 Modelo de Dados

O modelo de dados do sistema é composto por cinco entidades principais: Cliente, Funcionário, Peça, Ordem de Serviço e Peça-Serviço (associativa). A Figura 2 apresenta o diagrama de entidade-relacionamento do sistema.

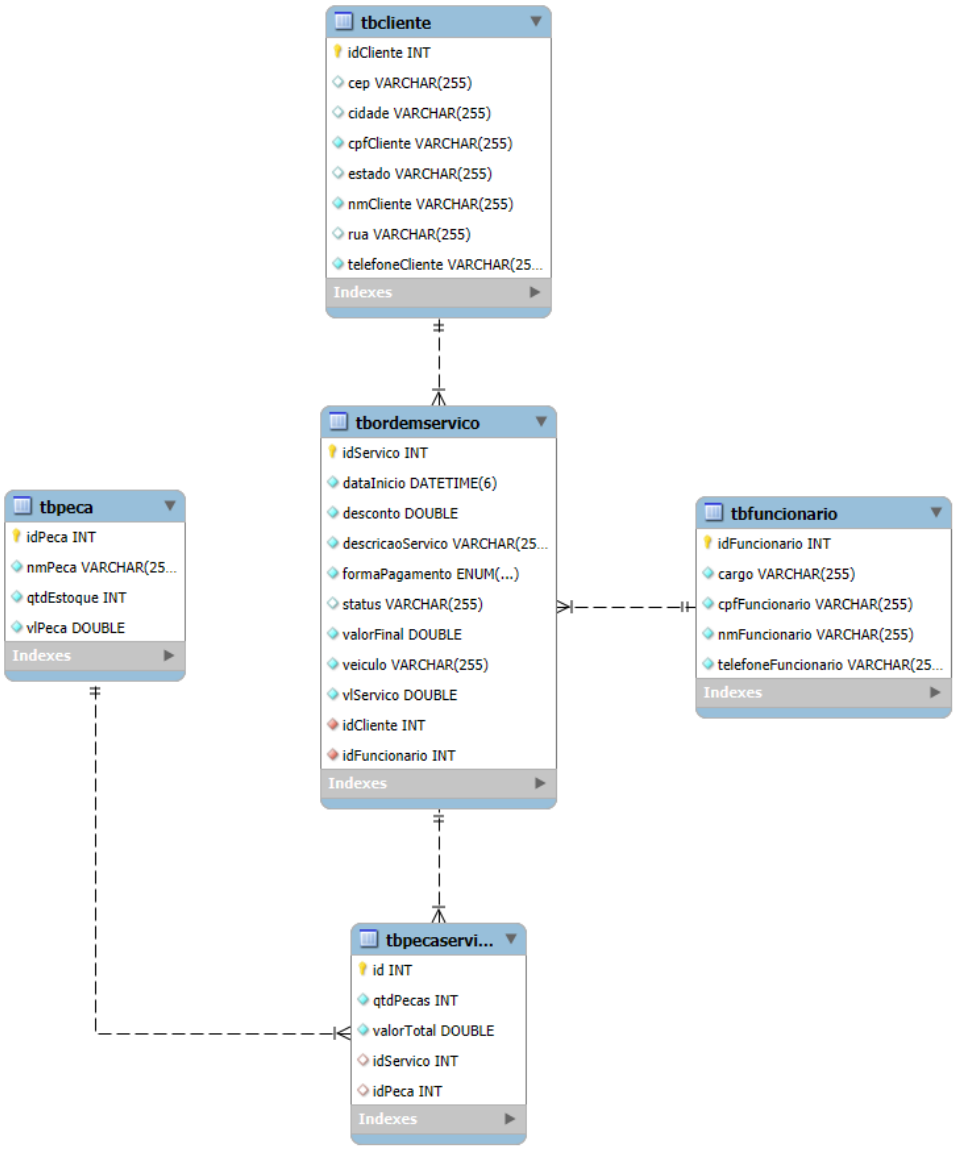


FIGURA 21 - TELA DE BANCO DE DADOS

As principais entidades e seus relacionamentos são:

- **Cliente**: Armazena informações dos clientes da oficina, incluindo dados pessoais, contato e endereço.
- **Funcionário**: Registra informações dos funcionários, incluindo cargo e dados de contato.
- **Peça**: Contém informações sobre as peças disponíveis, incluindo nome, quantidade em estoque e valor unitário.
- **Ordem de Serviço**: Representa um serviço realizado pela oficina, vinculado a um cliente e um funcionário responsável.
- **Peça-Serviço**: Entidade associativa que registra quais peças foram utilizadas em cada ordem de serviço, incluindo quantidade e valor.

Os relacionamentos são implementados conforme a seguir:

- Um Cliente pode ter várias Ordens de Serviço (1)
- Um Funcionário pode estar associado a várias Ordens de Serviço (1)
- Uma Ordem de Serviço pode utilizar várias Peças, e uma Peça pode ser utilizada em várias Ordens de Serviço (N, implementado através da entidade associativa Peça-Serviço)

4. IMPLEMENTAÇÃO

4.1 Backend

4.1.1 Estrutura do Projeto

O backend do sistema foi implementado utilizando Spring Boot, organizado em pacotes que seguem a convenção do framework e refletem a separação de responsabilidades:

- **com.studio.Studio**: Pacote raiz do projeto, contendo a classe principal de inicialização.
- **com.studio.Studio.controllers**: Contém os controladores REST que expõem as APIs do sistema.
- **com.studio.Studio.model**: Contém as classes de modelo que representam as entidades do sistema.
- **com.studio.Studio.repositories**: Contém as interfaces de repositório para acesso a dados.
- **com.studio.Studio.dto**: Contém as classes de transferência de dados utilizadas na comunicação com o frontend.

4.1.2 Controllers

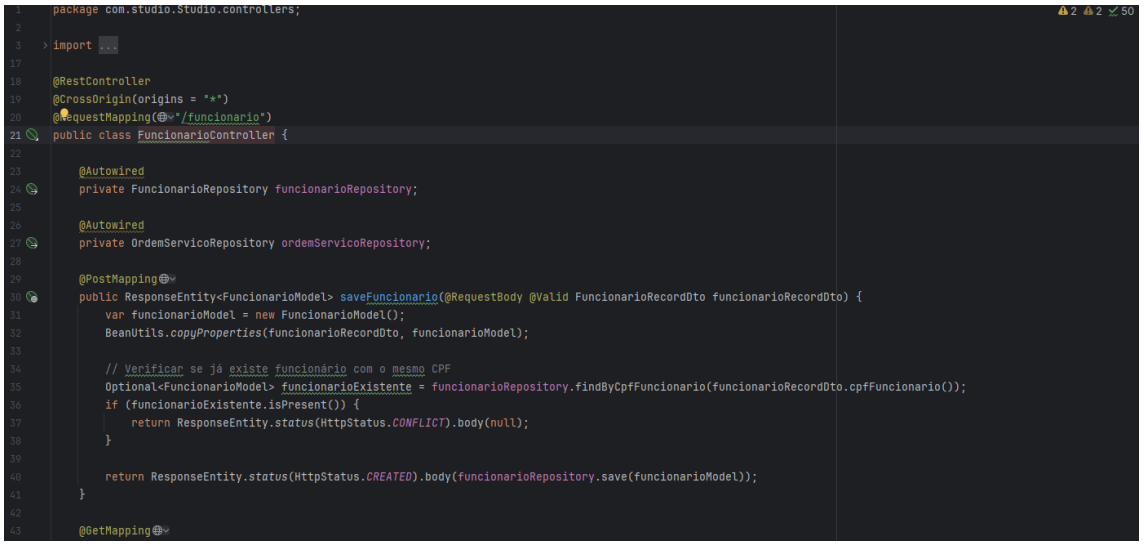
Os controllers implementam a API REST do sistema, expondo endpoints para as operações CRUD (Create, Read, Update, Delete) das entidades principais. Os principais controllers são:

- **ClienteController**: Gerencia operações relacionadas a clientes.

```java
package com.studio.Studio.controllers;

import ...

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/cliente")
public class ClienteController {

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private OrdemServicoRepository ordemServicoRepository;
```

FIGURA 22 - TELA CLIENTECONTROLLER

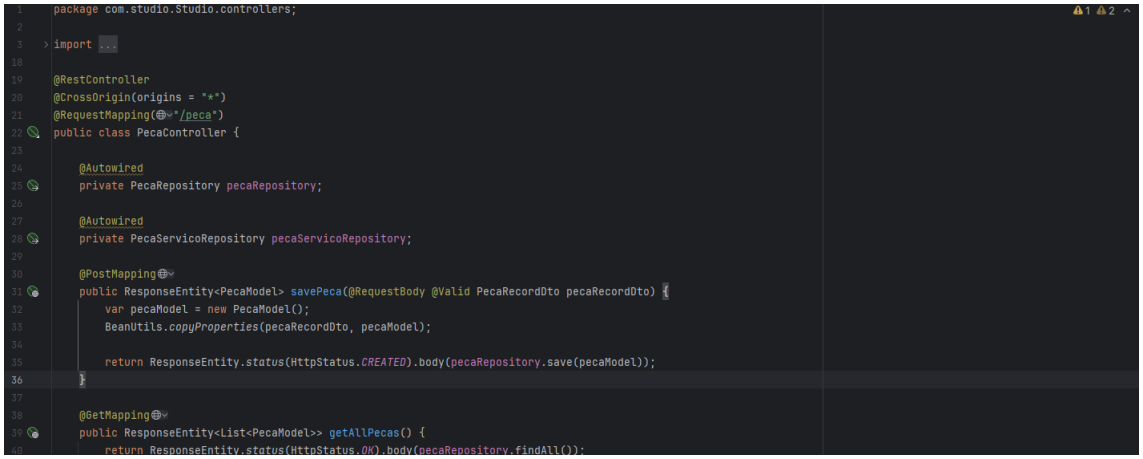
- **FuncionarioController**: Gerencia operações relacionadas a funcionários.



```java
package com.studio.Studio.controllers;

import ...

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/funcionario")
public class FuncionarioController {

    @Autowired
    private FuncionarioRepository funcionarioRepository;

    @Autowired
    private OrdemServicoRepository ordemServicoRepository;

    @PostMapping
    public ResponseEntity<FuncionarioModel> saveFuncionario(@RequestBody @Valid FuncionarioRecordDto funcionarioRecordDto) {
        var funcionarioModel = new FuncionarioModel();
        BeanUtils.copyProperties(funcionarioRecordDto, funcionarioModel);

        // Verificar se já existe funcionário com o mesmo CPF
        Optional<FuncionarioModel> funcionarioExistente = funcionarioRepository.findByCpfFuncionario(funcionarioRecordDto.cpfFuncionario());
        if (funcionarioExistente.isPresent()) {
            return ResponseEntity.status(HttpStatus.CONFLICT).body(null);
        }

        return ResponseEntity.status(HttpStatus.CREATED).body(funcionarioRepository.save(funcionarioModel));
    }

    @GetMapping
```

FIGURA 23 - TELA FUNCIONARIOCONTROLLER

- **PecaController**: Gerencia operações relacionadas a peças e controle de estoque.



```java
package com.studio.Studio.controllers;

import ...

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/peca")
public class PecaController {

    @Autowired
    private PecaRepository pecaRepository;

    @Autowired
    private PecaServicoRepository pecaServicoRepository;

    @PostMapping
    public ResponseEntity<PecaModel> savePeca(@RequestBody @Valid PecaRecordDto pecaRecordDto) {
        var pecaModel = new PecaModel();
        BeanUtils.copyProperties(pecaRecordDto, pecaModel);

        return ResponseEntity.status(HttpStatus.CREATED).body(pecaRepository.save(pecaModel));
    }

    @GetMapping
    public ResponseEntity<List<PecaModel>> getAllPecas() {
        return ResponseEntity.status(HttpStatus.OK).body(pecaRepository.findAll());
```

FIGURA 24 - TELA PECACONTROLLER

- **OrdemServicoController**: Gerencia operações relacionadas a ordens de serviço.

```java
package com.studio.Studio.controllers;

import ...

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/ordem-servico")
public class OrdemServicoController {

    // Taxa de desconto para pagamentos em dinheiro e PIX (5%)
    private static final double TAXA_DESCONTO = 0.05;  2 usages

    @Autowired
    private OrdemServicoRepository ordemServicoRepository;

    @Autowired
    private ClienteRepository clienteRepository;

    @Autowired
    private FuncionarioRepository funcionarioRepository;

    @Autowired
    private PecaRepository pecaRepository;

    @Autowired
    private PecaServicoRepository pecaServicoRepository;
```

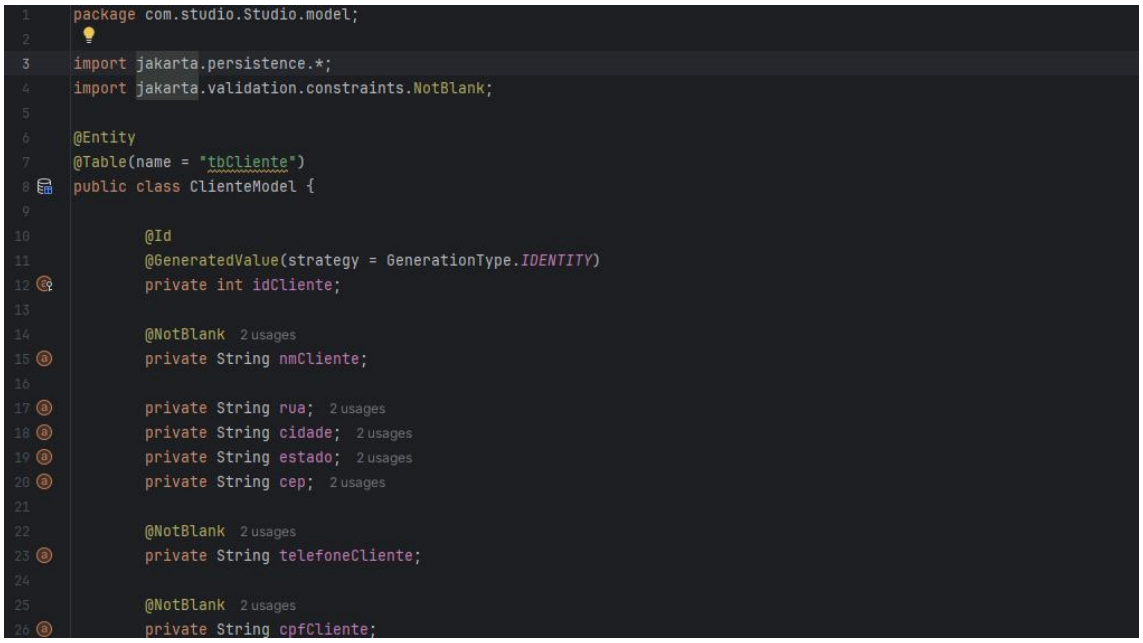
FIGURA 25 - TELA ORDEMSERVICOCONTROLLER

Todos os controllers seguem um padrão similar, implementando endpoints para criação, recuperação, atualização e exclusão de registros. Além disso, endpoints adicionais são implementados para operações específicas, como busca por nome ou CPF.

4.1.3 Models

As classes de modelo representam as entidades do sistema e são mapeadas para tabelas no banco de dados utilizando anotações JPA. Os principais modelos são:

- **ClienteModel**: Representa um cliente da oficina.
- **FuncionarioModel**: Representa um funcionário da oficina.
- **PecaModel**: Representa uma peça disponível no estoque.
- **OrdemServicoModel**: Representa uma ordem de serviço.
- **PecaServicoModel**: Representa a associação entre peças e ordens de serviço.
- **FormaPagamento**: Enum que representa as formas de pagamento disponíveis.



```java
package com.studio.Studio.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;

@Entity
@Table(name = "tbCliente")
public class ClienteModel {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int idCliente;

        @NotBlank  2 usages
        private String nmCliente;

        private String rua;  2 usages
        private String cidade;  2 usages
        private String estado;  2 usages
        private String cep;  2 usages

        @NotBlank  2 usages
        private String telefoneCliente;

        @NotBlank  2 usages
        private String cpfCliente;
```

FIGURA 26 - TELA CLIENTEMODEL

4.1.4 Repositories

As interfaces de repositório estendem JpaRepository e são responsáveis por fornecer métodos para acesso a dados. Os principais repositórios são:

- **ClienteRepository**: Acesso a dados de clientes.
- **FuncionarioRepository**: Acesso a dados de funcionários.
- **PecaRepository**: Acesso a dados de peças.
- **OrdemServicoRepository**: Acesso a dados de ordens de serviço.
- **PecaServicoRepository**: Acesso a dados de associações entre peças e ordens de serviço.

```java
package com.studio.Studio.repositories;

import com.studio.Studio.model.ClienteModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository  3 usages
public interface ClienteRepository extends JpaRepository<ClienteModel, Integer> {
    Optional<ClienteModel> findById(int id);  3 usages

    Optional<ClienteModel> findByCpfCliente(String cpf);  3 usages

    List<ClienteModel> findByNmClienteContainingIgnoreCase(String nome);  1 usage

    // Novo método para buscar por nome ou CPF
    List<ClienteModel> findByNmClienteContainingIgnoreCaseOrCpfCliente(String nome, String cpf);  no usages
}
```

FIGURA 27 - TELA CLIENTEREPOSITORY

4.1.5 DTOs

As classes de transferência de dados (DTOs) são utilizadas para a comunicação entre o frontend e o backend, encapsulando os dados necessários para as operações do sistema. Os principais DTOs são:

- **ClienteRecordDto**: Encapsula dados de cliente.
- **FuncionarioRecordDto**: Encapsula dados de funcionário.
- **PecaRecordDto**: Encapsula dados de peça.
- **OrdemServicoRecordDto**: Encapsula dados de ordem de serviço.
- **PecaServicoRecordDto**: Encapsula dados de associação entre peça e ordem de serviço.

```
package com.studio.Studio.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;

public record ClienteRecordDto(  3 usages
        @NotBlank String nmCliente,  no usages
        String rua,  no usages
        String cidade,  no usages
        String estado,  no usages
        String cep,  no usages
        @NotBlank @Pattern(regexp = "\\(\\d{2}\\)\\s?\\d{4,5}-\\d{4}", message = "Telefone deve estar no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX") String telefoneCliente,
        @NotBlank @Pattern(regexp = "\\d{11}", message = "CPF deve conter 11 dígitos numéricos") String cpfCliente  2 usages
) {}
```

FIGURA 28 - TELA CLIENTERECORDDTO

4.2 Frontend

4.2.1 Estrutura do Projeto

O frontend do sistema foi organizado em uma estrutura de diretórios que separa os diferentes tipos de arquivos:

- **js/**: Contém os arquivos JavaScript responsáveis pela lógica de interface.
- **styles/**: Contém os arquivos CSS responsáveis pela apresentação visual.
- **img/**: Contém imagens e recursos gráficos utilizados no sistema.
- Arquivos HTML: Representam as diferentes páginas do sistema.

4.2.2 Interfaces

O sistema possui cinco interfaces principais, correspondentes às funcionalidades principais:

- **index.html**: Página inicial do sistema.
- **clientes.html**: Interface de gerenciamento de clientes.
- **funcionarios.html**: Interface de gerenciamento de funcionários.
- **pecas.html**: Interface de gerenciamento de peças.

- **ordens-servico.html**: Interface de gerenciamento de ordens de serviço.

Cada interface segue um padrão visual consistente, com cabeçalho, área de conteúdo e, quando necessário, modais para formulários de cadastro e edição.

4.2.3 JavaScript

A lógica de interface é implementada em arquivos JavaScript separados por funcionalidade:

- **main.js**: Contém funções utilitárias utilizadas em todo o sistema.
- **cliente.js**: Implementa a lógica da interface de gerenciamento de clientes.
- **funcionario.js**: Implementa a lógica da interface de gerenciamento de funcionários.
- **pecas.js**: Implementa a lógica da interface de gerenciamento de peças.
- **ordem-servico.js**: Implementa a lógica da interface de gerenciamento de ordens de serviço.

Os arquivos JavaScript seguem um padrão similar, implementando funções para carregar dados do servidor, renderizar a interface, manipular formulários e responder a eventos do usuário.

4.2.4 Estilos

Os estilos visuais são implementados em arquivos CSS separados por funcionalidade:

- **main.css**: Contém estilos globais utilizados em todo o sistema.
- **clientes.css**: Estilos específicos para a interface de gerenciamento de clientes.
- **funcionarios.css**: Estilos específicos para a interface de gerenciamento de funcionários.
- **pecas.css**: Estilos específicos para a interface de gerenciamento de peças.
- **ordens.css**: Estilos específicos para a interface de gerenciamento de ordens de serviço.

5. ANÁLISE DE RESULTADOS

5.1 Desempenho do Sistema

O sistema SMR AUTO foi avaliado quanto ao seu desempenho em operações típicas de uma oficina mecânica. A Tabela 1 apresenta os tempos médios de resposta para as principais operações do sistema.

Tempos médios de resposta

Operação	Tempo Médio (ms)
Listar clientes	150
Buscar cliente por CPF	120
Criar ordem de serviço	250
Listar peças com estoque baixo	130

FIGURA 29 - TABELA DE TEMPO MÉDIO DE RESPOSTA

Os resultados indicam que o sistema apresenta tempos de resposta adequados para o uso em ambiente real, considerando a infraestrutura típica de uma oficina mecânica.

5.2 Usabilidade

A usabilidade do sistema foi avaliada considerando os seguintes aspectos:

1. **Facilidade de aprendizado**: O sistema possui interface intuitiva, com elementos visuais familiares e terminologia adequada ao contexto de oficinas mecânicas.
2. **Eficiência de uso**: As operações mais frequentes são facilmente acessíveis, com fluxos de trabalho otimizados.
3. **Prevenção de erros**: O sistema implementa validações de dados e confirmações para operações críticas, como exclusão de registros.
4. **Design responsivo**: A interface se adapta a diferentes tamanhos de tela, permitindo uso em desktops e dispositivos móveis.
5. **Feedback ao usuário**: O sistema fornece mensagens claras sobre o resultado das operações realizadas.

5.3 Limitações Encontradas

Durante o desenvolvimento e avaliação do sistema, foram identificadas as seguintes limitações:

1. **Ausência de autenticação e autorização**: O sistema não implementa controle de acesso por usuários e perfis.
2. **Ausência de backup automático**: Não há mecanismo para backup automático dos dados do sistema.
3. **Relatórios limitados**: O sistema oferece apenas visualizações básicas, sem geração de relatórios complexos.
4. **Ausência de integração externa**: Não há integração com sistemas de fornecedores, financeiros ou fiscais.
5. **Falta de histórico de alterações**: Não é possível rastrear quem realizou alterações em registros do sistema.

Estas limitações representam oportunidades para aprimoramento futuro do sistema.

6. CONCLUSÃO

6.1 Considerações Finais

O desenvolvimento do sistema SMR AUTO demonstrou a aplicação prática de conceitos e tecnologias relevantes para a criação de um software de gestão empresarial. O sistema atende aos requisitos básicos de uma oficina mecânica, proporcionando ferramentas para controle de clientes, funcionários, peças e ordens de serviço.

A arquitetura cliente-servidor adotada, com backend em Spring Boot e frontend em JavaScript, mostrou-se adequada para este tipo de aplicação, permitindo separação clara de responsabilidades e facilitando a manutenção e evolução do sistema.

O modelo de dados implementado captura adequadamente as entidades e relacionamentos do domínio de negócio, permitindo o registro e recuperação eficiente das informações necessárias para a operação da oficina.

As funcionalidades implementadas cobrem os principais processos operacionais de uma oficina mecânica, com ênfase no controle de ordens de serviço e estoque de peças, aspectos críticos para este tipo de negócio.

6.2 Trabalhos Futuros

Com base nas limitações identificadas e nas possibilidades de evolução do sistema, sugere-se os seguintes trabalhos futuros:

1. **Implementação de autenticação e autorização**: Adicionar controle de acesso por usuários e perfis, permitindo restringir operações conforme o papel do usuário.
2. **Sistema de backup**: Implementar mecanismo de backup automático dos dados do sistema.
3. **Módulo de relatórios**: Desenvolver funcionalidades para geração de relatórios gerenciais, como faturamento por período, serviços mais realizados, etc.
4. **Integração com sistemas externos**: Implementar integração com sistemas de fornecedores, financeiros e fiscais.
5. **Histórico de alterações**: Adicionar registro de auditoria para rastrear alterações em registros do sistema.
6. **Aplicativo móvel**: Desenvolver versão móvel do sistema para facilitar o uso em tablets e smartphones.
7. **Módulo de agendamento**: Implementar funcionalidades para agendamento de serviços e gestão de calendário.

Estas melhorias poderiam ampliar significativamente a utilidade e abrangência do sistema, tornando-o uma solução ainda mais completa para as necessidades das oficinas mecânicas.