Documentação Completa: Sistema de Gerenciamento de Oficina Mecânica

Sumário

14. **Testes**

15. **Manutenção e Suporte**

---

1. Introdução

O **Sistema de Gerenciamento de Oficina Mecânica** foi desenvolvido para proporcionar uma plataforma robusta e eficiente para o gerenciamento de oficinas de manutenção automotiva. Ele tem como foco a organização de processos internos, controle de agendamentos, gestão de serviços, peças, veículos e clientes, além da geração de relatórios financeiros e operacionais. A aplicação é baseada no framework **Django**, usando tecnologias modernas para o front-end como **React.js** e **Tailwind CSS**, visando garantir uma experiência de usuário fluida e responsiva.

O sistema permite o controle de agendamentos, gerenciamento de clientes e veículos, manutenção de estoque, emissão de orçamentos e faturas, e geração de relatórios para a gestão da oficina.

---

2. Objetivos

O principal objetivo deste projeto é automatizar o gerenciamento de processos em uma oficina mecânica, tornando a operação mais eficiente, reduzindo erros manuais e melhorando a experiência tanto para os atendentes quanto para os clientes. O sistema visa:

- **Melhorar o controle e a organização de dados** como clientes, veículos, serviços, e peças.
- **Reduzir falhas humanas** ao automatizar processos como agendamento e emissão de faturas.
- **Facilitar a comunicação entre os diferentes papéis** dentro da oficina, como mecânicos, atendentes e administradores.
- **Fornecer relatórios claros e precisos**, permitindo uma gestão eficaz da oficina.

---

3. Requisitos

3.1 Requisitos Funcionais

- **Cadastro de Clientes**: O sistema permite registrar e visualizar dados de clientes, incluindo nome, telefone, e-mail e endereço.

- **Cadastro de Veículos**: Associar veículos aos clientes, com dados como modelo, placa, tipo (carro, moto, caminhão), e ano de fabricação.

- **Agendamento de Serviços**: Permite a criação de serviços de manutenção com datas e horários, associando peças e mecânicos responsáveis.

- **Controle de Serviços**: O sistema permite registrar, editar e acompanhar os status dos serviços (pendente, em andamento, concluído).

- **Controle de Estoque de Peças**: Acompanhamento de peças, quantidades disponíveis, preço, e fornecedor, com atualização automática ao usar peças nos serviços.

- **Orçamentos e Faturas**: Geração de orçamentos detalhados com base nos serviços agendados e peças utilizadas, além da emissão de faturas ao término dos serviços.

- **Relatórios**: O sistema gera relatórios financeiros (faturamento, vendas) e operacionais (serviços realizados, peças consumidas).

3.2 Requisitos Não Funcionais

- **Desempenho**: O sistema precisa ser capaz de lidar com grandes volumes de dados e múltiplos usuários simultâneos sem comprometer a performance.

- **Segurança**: A proteção de dados sensíveis dos clientes e operações financeiras é crucial, utilizando criptografia e autenticação forte.

- **Escalabilidade**: A solução deve permitir adicionar mais funcionalidades ou expandir para várias filiais de forma simples.

- **Usabilidade**: A interface deve ser intuitiva, com fluxos claros e eficientes para os usuários.

---

4. Tecnologias Utilizadas

Back-End

- **Django**: Framework web Python para desenvolvimento de back-end robusto e escalável.

- **Django REST Framework**: Para construir a API RESTful que será consumida pelo front-end.
- **PostgreSQL**: Banco de dados relacional para persistir dados de clientes, veículos, serviços, peças e outros.
- **Celery**: Ferramenta para processamento assíncrono de tarefas, como o envio de e-mails ou a geração de relatórios.
- **JWT (JSON Web Tokens)**: Para a autenticação e autorização segura dos usuários.

Front-End

- **React.js**: Biblioteca JavaScript para criar interfaces de usuário dinâmicas e reativas.
- **Tailwind CSS**: Framework CSS para facilitar o design responsivo e modular.
- **Axios**: Biblioteca para realizar requisições HTTP de maneira eficiente ao back-end.

Segurança

- **HTTPS**: Garantia de comunicação segura entre o cliente e o servidor.
- **Autenticação via JWT**: Para garantir que apenas usuários autenticados acessem as informações sensíveis.

---

5. Arquitetura do Sistema

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

Camadas do Sistema

1. **Camada de Apresentação (Front-End)**: Responsável pela interação com o usuário. Utiliza **React.js** e **Tailwind CSS** para criar uma interface dinâmica e responsiva.
2. **Camada de Lógica de Negócio (Back-End)**: Responsável pelo processamento dos dados e pela comunicação com o banco de dados, utilizando **Django** e **Django REST Framework**.

3. **Camada de Persistência de Dados (Banco de Dados)**: Utiliza **PostgreSQL** para armazenar e organizar dados de clientes, veículos, serviços, peças, etc.

Fluxo de Dados

- O **Front-End** envia requisições HTTP para a **API** (back-end).
- O **Back-End** processa as requisições e interage com o **Banco de Dados** para armazenar ou recuperar informações.
- O **Front-End** recebe os dados e os exibe ao usuário.

---

6. Estrutura de Diretórios

A estrutura de diretórios do projeto é organizada para facilitar a navegação e manutenção. A estrutura básica é:

/gerenciamento_oficina

```
├── /config/              # Arquivos de configuração do Django
├── /clientes/            # App de gerenciamento de clientes
├── /veiculos/            # App de gerenciamento de veículos
├── /servicos/            # App de gerenciamento de serviços
├── /pecas/               # App de controle de peças e estoque
├── /relatorios/          # App para geração de relatórios
├── /templates/           # Templates HTML (para views do Django)
├── /static/              # Arquivos estáticos (CSS, JS, imagens)
├── /media/               # Arquivos de mídia (ex: imagens de clientes ou veículos)
├── /migrations/          # Migrações do banco de dados
├── /tests/               # Testes automatizados
└── manage.py             # Script de gerenciamento
```

---

7. Banco de Dados

Estrutura das Tabelas

1. **Cliente**: Contém informações como nome, telefone, e-mail e endereço.
2. **Veículo**: Associa veículos aos clientes com dados como modelo, placa e ano.
3. **Serviço**: Registra serviços realizados, com status e valor.
4. **Peça**: Detalha as peças no estoque, incluindo descrição, quantidade e preço.
5. **Orçamento**: Registra orçamentos associados a serviços.

---

8. Funcionalidades do Sistema

Front-End

- **Interface de Clientes**: Cadastro e edição de clientes e veículos.
- **Agenda de Serviços**: Visualização de serviços agendados e execução de ações.
- **Feedback Visual**: Exibição de mensagens de sucesso ou erro em tempo real.

Back-End

- **Autenticação de Usuários**: Proteção de rotas e dados, utilizando **JWT** para login e autorização.
- **Gestão de Dados**: Acesso e manipulação de dados de clientes, veículos, serviços e estoque.
- **API RESTful**: Comunicação entre o front-end e back-end via endpoints.

---

9. Usuários do Sistema

1. Administrador

Responsável por gerenciar o sistema, cadastrar usuários e visualizar relatórios financeiros e operacionais. Tem permissões completas sobre todos os dados e funcionalidades.

2. Atendente

Cadastra e gerencia clientes, veículos e agendamentos de serviços. Possui permissões limitadas para editar dados de clientes e gerar orçamentos.

3. Mecânico

Responsável por executar serviços e registrar o uso de peças. Possui permissões para alterar o status de serviços, mas não pode acessar ou modificar dados financeiros.

4. Supervisor de Estoque

Gerencia o estoque de peças, fazendo a adição e remoção de itens, e gerando relatórios sobre a disponibilidade de peças.

---

10. Fluxo de Trabalho

1. **Login**: O usuário faz login usando credenciais seguras.

2. **Cadastro e Agendamento**:

O atendente cadastra clientes, veículos e agenda serviços.
3. **Execução do Serviço**: O mecânico realiza o serviço e atualiza o status.
4. **Geração de Fatura**: O atendente ou administrador gera faturas e orçamentos.

---

11. Segurança e Autenticação

- **Autenticação JWT**: Para garantir que apenas usuários autenticados possam acessar o sistema.

- **Criptografia de Dados Sensíveis**: Dados como senhas e informações financeiras são criptografados para garantir a segurança.

- **Controle de Acesso por Função**: Diferentes papéis de usuário possuem diferentes níveis de acesso e permissões.

---

12. Design de Interface (UI) e Experiência do Usuário (UX)

A interface foi projetada com foco na **simplicidade** e **acessibilidade**. O uso de **Tailwind CSS** proporciona um design moderno e responsivo, com navegação fácil e intuitiva.

---

13. Implantação

O sistema pode ser implantado em servidores **cloud** ou servidores locais. A implementação de **Docker** para conteinerização permite facilidade na instalação e escalabilidade.

---

14. Testes

O sistema foi testado com **unit tests** para garantir a funcionalidade correta de cada componente. **Testes de integração** foram realizados para garantir que os diferentes módulos funcionem de forma integrada.

---

15. Manutenção e Suporte

A manutenção do sistema envolve:

- **Atualizações periódicas** para melhorar a performance e segurança.
- **Monitoramento contínuo** para identificar e corrigir bugs ou falhas de segurança.
