|  |  |
| --- | --- |
| Sumário  1. INTRODUÇÃO |  |
| 2. OBJETIVOS |  |
| 3. FUNDAMENTAÇÃO TEÓRICA |  |
| 4. METODOLOGIA |  |
| 4.1. ARQUITETURA DO SISTEMA |  |
| 4.2. FUNCIONALIDADES |  |
| 5. RESULTADOS |  |
| 6. CONSIDERAÇÕES FINAIS |  |
| **7. APÊNDICE A — ESTRUTURA DOS PRINCIPAIS ARQUIVOS** |  |
| **8. APÊNDICE B — DIAGRAMA SIMPLIFICADO DE NAVEGAÇÃO** |  |

**1. INTRODUÇÃO**

A necessidade de sistemas informatizados eficientes é crescente em diversos setores, incluindo o ramo de vistorias veiculares. O presente projeto propõe uma solução completa que integra o gerenciamento de clientes, veículos, agendamentos, vistorias e pagamentos, contemplando diferentes perfis de usuário: vistoriador, gerente e cliente.

**2. OBJETIVOS**

O objetivo geral deste projeto é desenvolver um sistema robusto para o gerenciamento de vistorias veiculares.

Os objetivos específicos são:

* Automatizar o processo de agendamento e a realização das vistorias.
* Permitir o acompanhamento do *status* das vistorias e dos pagamentos em tempo real.
* Facilitar a gestão de funcionários, clientes e veículos.
* Proporcionar relatórios e *dashboards* customizáveis para apoio à tomada de decisão.

**3. FUNDAMENTAÇÃO TEÓRICA**

O sistema utiliza o padrão arquitetural MVC (*Model-View-Controller*) para separar as responsabilidades do código, facilitando a manutenção e a escalabilidade. A interface gráfica (*View*) é desenvolvida com Java Swing, proporcionando uma experiência gráfica amigável e responsiva ao usuário. A camada de persistência de dados utiliza DAOs (*Data Access Objects*), o que permite a abstração do acesso aos dados, promovendo flexibilidade para futuras integrações com diferentes sistemas gerenciadores de banco de dados.

**4. METODOLOGIA**

**4.1. ARQUITETURA DO SISTEMA**

O sistema foi concebido em uma arquitetura em camadas, seguindo o padrão MVC:

* **Model**: Camada que contém as classes que representam as entidades de negócio (e.g., Cliente, Veículo, Funcionario, Agendamento, Vistoria).
* **DAO** (*Data Access Object*): Camada responsável por abstrair as operações de persistência e recuperação de dados.
* **Controller**: Camada que centraliza as regras de negócio e manipula os dados entre a Model e a View.
* **View**: Camada que implementa a interface gráfica, composta por três *dashboards* principais para os diferentes perfis de usuário:
  + DashboardVistoriador: Painel para o vistoriador, gerenciando agendamentos, vistorias e relatórios operacionais.
  + DashBoardGerente: Painel para o gerente, focado na gestão de funcionários e em relatórios gerenciais.
  + DashBoardCliente: Painel para o cliente, permitindo visualização de dados pessoais, agendamentos e pagamentos.

**4.2. FUNCIONALIDADES**

**DashboardVistoriador**

* Visualização de agendamentos pendentes e concluídos.
* Cadastro de novos clientes, veículos e agendamentos.
* Registro detalhado de vistorias, incluindo *checklist* e observações.
* Relatório detalhado de vistorias com filtros por data, cliente e funcionário.
* Exportação de relatórios em formato CSV.

**DashBoardGerente**

* Visualização e gerenciamento de funcionários (cadastro, edição, exclusão).
* Resumo estatístico do desempenho de funcionários.
* Relatório gerencial (em construção - *feature* a ser implementada na fase 2).

**DashBoardCliente**

* Visualização e edição de dados pessoais.
* Consulta de agendamentos e *status* de pagamentos.
* Pagamento *online* de vistorias realizadas.
* Visualização de relatórios das vistorias efetuadas.

**5. RESULTADOS**

O sistema desenvolvido proporciona os seguintes resultados:

* **Gestão Centralizada**: Os processos de vistoria são gerenciados de forma unificada.
* **Melhoria Operacional**: Redução de retrabalhos e erros humanos por meio da automação.
* **Usabilidade**: Facilidade de uso garantida pela interface Swing, adaptada aos diferentes perfis.
* **Suporte à Decisão**: Geração de relatórios customizáveis que apoiam a tomada de decisão gerencial.

**6. CONSIDERAÇÕES FINAIS**

O projeto atende satisfatoriamente aos requisitos funcionais primários de uma empresa de vistoria veicular. Para futuras expansões, sugere-se a integração com um banco de dados relacional (e.g., MySQL, PostgreSQL), o desenvolvimento de funcionalidades complementares (como notificações automáticas por e-mail/SMS) e a implementação de um controle de acesso mais refinado por níveis de permissão.

**7. APÊNDICE A — ESTRUTURA DOS PRINCIPAIS ARQUIVOS**

* view/DashboardVistoriador.java: Implementação da interface gráfica principal para vistoriadores, abrangendo funcionalidades de cadastro, registro de vistoria e relatórios.
* view/DashBoardGerente.java: Implementação da interface gráfica principal para o gerente, com foco no gerenciamento de funcionários.
* view/DashBoardCliente.java: Implementação da interface gráfica para o cliente, oferecendo acesso a dados pessoais, agendamentos e relatórios de vistorias.
* model/Cliente.java: Classe que representa a entidade cliente.
* dao/ClienteDAO.java: Classe que implementa as operações de persistência para a entidade cliente.
* controller/AgendamentoController.java: Classe que gerencia a lógica de negócio de agendamentos.