Relatório Final: Sistema de Gestão de Oficina Mecânica

1. Contribuição Individual (O que o aluno fez no projeto)

João Matheus Ramos Araujo:

- Responsável pela implementação de controllers importantes como o de criar ordem de serviços, os detalhes de ordem de serviço, histórico veicular e veículos.
- Desenvolveu as interfaces de ordens de serviços, criar ordens de serviços, veículos e histórico veicular.
- Trabalhou na lógica de validação de dados de CPF, email e telefone.

Erick Rhuan Carvalho de Freitas Pereira:

- Focou-se no desenvolvimento dos Controllers restantes e na integração entre Back-end e Front-end.
- Responsável pela gestão de dependências e estrutura do projeto.
- **Implementação dos Controllers e interfaces mais básicas**(como a Gestão de Clientes e a Gestão de Estoque/Peças).

Igor Pereira Lima:

- Segundo responsável pela revisão da lógica de negócio na parte de Gestão de Estoque, Gestão de Peças, Manutenção e Ordem de Serviço.
- Implementou a lógica de Query no banco de dados, aplicando os padrões de projeto de Data Access Object (DAO).
- Confecção do diagrama UML de casos de uso

---

2. Funcionalidades Implementadas

- **Autenticação e Segurança:**
  - Sistema de Login para administradores, garantindo acesso restrito às funcionalidades de gestão.
  - Implementação de boas práticas de segurança através do uso de variáveis de ambiente (`.env`) para ocultar credenciais da base de dados.
- **Gestão de Clientes e Veículos (CRUD Completo):**
  - Registo, edição, listagem e remoção de clientes com validação de dados (CPF e telefone).
  - Associação direta de veículos aos seus proprietários, permitindo histórico de manutenção individualizado.
- **Gestão de Ordens de Serviço (Core do Sistema):**

    - Criação de Ordens de Serviço (OS) detalhadas, incluindo veículo, descrição do problema e mão de obra.
    - **Gestão de Estado:** Controle do fluxo da OS através de estados ("Em Serviço", "Aguardando Peças", "Pronto", "Finalizado").
    - **Integração com estoque:** Adição dinâmica de peças à OS, com cálculo automático do valor total e baixa automática no estoque.
- **Gestão de Estoque e Peças:**
    - Controle de inventário com funcionalidades de entrada e saída de material.
    - Sistema de alertas visuais para peças com estoque baixo (inferior a 10 unidades).
- **Módulo Financeiro e Relatórios:**
    - Registo de pagamentos com múltiplas formas (Dinheiro, Pix, Cartão).
    - Geração automática de relatórios financeiros em formato **PDF**, detalhando o faturamento e as ordens finalizadas, prontos para exportação.

---

3. Funcionalidades Não Implementadas e Justificação*.*

As funcionalidades de **Agendamento de Serviços** e **Sistema de Clientes VIP** não foram implementadas. Esta decisão foi tomada com base nos seguintes fatores:

1. **Escala e Complexidade do Projeto:** Durante o desenvolvimento, o núcleo do sistema (gestão de OS e integração com estoque/financeiro) revelou-se complexo e robusto. A equipa decidiu focar os esforços em refinar estas funcionalidades críticas para garantir que não existissem falhas no fluxo principal de trabalho (do orçamento à faturação).
2. **Gestão de Tempo e Prioridades:** Dado o prazo de entrega, a implementação de um calendário de agendamentos exigiria uma lógica temporal complexa que poderia comprometer a estabilidade das funcionalidades já existentes.
3. **Reavaliação de Necessidade:** Para o âmbito deste projeto, considerou-se que o controlo de estado das Ordens de Serviço ("Aguardando", "Em Serviço") já satisfaz a necessidade de organização do fluxo de trabalho, tornando o agendamento prévio uma funcionalidade secundária neste momento.

---

4. Experiência do Aluno

preenchimento individual:

Aspetos Positivos:

- Aprender a gerir conexões JDBC e evitar SQL injections foi uma grande passo na minha visão para aprender mais sobre como o java funciona e sua efetividade. (João Matheus)
- Aprimorei o trabalho em equipe, dividindo tarefas e funções de cada membro.(Igor Pereira Lima)
- Aprendi mais sobre o padrão de projeto DAO.(Igor Pereira Lima)

- Aprimorou a capacidade de construir interfaces de usuário claras (JavaFX/FXML) e fortaleceu a habilidade de integrar a lógica do Front-end com as funcionalidades do Back-end de Controllers.(Erick Rhuan)

Maiores Dificuldades:

- Na classe Criar Ordem De Serviço Controller, houve muita dificuldade em relação aos valores monetários, não estávamos conseguindo gerenciar esse fator corretamente. (João Matheus)
- Pensar nas classes do sistema e como ficariam com o padrão de projeto DAO (Igor Pereira Lima)
- "A gestão de dependências externas e a configuração das variáveis de ambiente para segurança da base de dados exigiram bastante pesquisa."(João Matheus)
- Enfrentou desafios na gestão de dependências e na garantia da estabilidade das interfaces mais básicas (como a Gestão de Clientes e Peças) ao integrá-las ao fluxo principal do sistema.(Erick Rhuan)

Aprendizagem:

- Consegui aplicar alguns conceitos vistos nas aulas de POO, isso só já se trata de uma grande avanço na minha habilidade como programador, além disso o uso do scene builder para construir interfaces facilitou muito a aprendizagem. (João Matheus)
- Tive uma grande evolução na parte de padrões de projeto e arquitetura de software ao operar Query para o DAO e separar as classes em no modelo MVC. Também consegui entender melhor esses mesmos conceitos apresentados pelo professor. Além do mais, a oportunidade de trabalhar em equipe foi em excelente ponto, porque a divisão de tarefas facilita o desenvolvimento, além de ter a noção de quanto tempo leva para implementar e testar cada parte do software. (Igor Pereira Lima)
- Obteve um melhor entendimento da arquitetura de software (MVC) ao separar a lógica de negócio nos Controllers. Além disso, a separação entre a lógica (Java) e o layout (FXML) facilitou a organização do código e a manutenção da interface. (Erick Rhuan Carvalho de Freitas Pereira)

---

5. Bibliotecas Utilizadas e Motivação Tecnológica

O projeto foi desenvolvido em **Java 21**, utilizando as seguintes tecnologias chave:

1. **JavaFX (Interface Gráfica):**
    - *Motivo:* Escolhido pela sua capacidade de criar interfaces modernas e ricas em desktop. A separação entre a lógica (Java) e o layout (FXML) facilitou a organização do código e a manutenção da interface.
2. **MySQL Connector (JDBC):**
    - *Motivo:* Essencial para a persistência de dados. O MySQL é um sistema de gestão de bases de dados relacional robusto e amplamente utilizado no mercado, ideal para gerir as relações entre Clientes, Veículos e Ordens de Serviço.
3. **iTextPDF:**

    - *Motivo:* Utilizada para a geração profissional de documentos. Permitiu criar relatórios financeiros estáticos e formatados, algo que seria extremamente complexo de fazer manualmente apenas com Java.
4. **Dotenv-java (Recomendado):**
    - *Motivo:* Implementado para garantir a segurança do projeto, evitando que credenciais sensíveis da base de dados ficassem expostas no código fonte.

---

6. Referências Consultadas

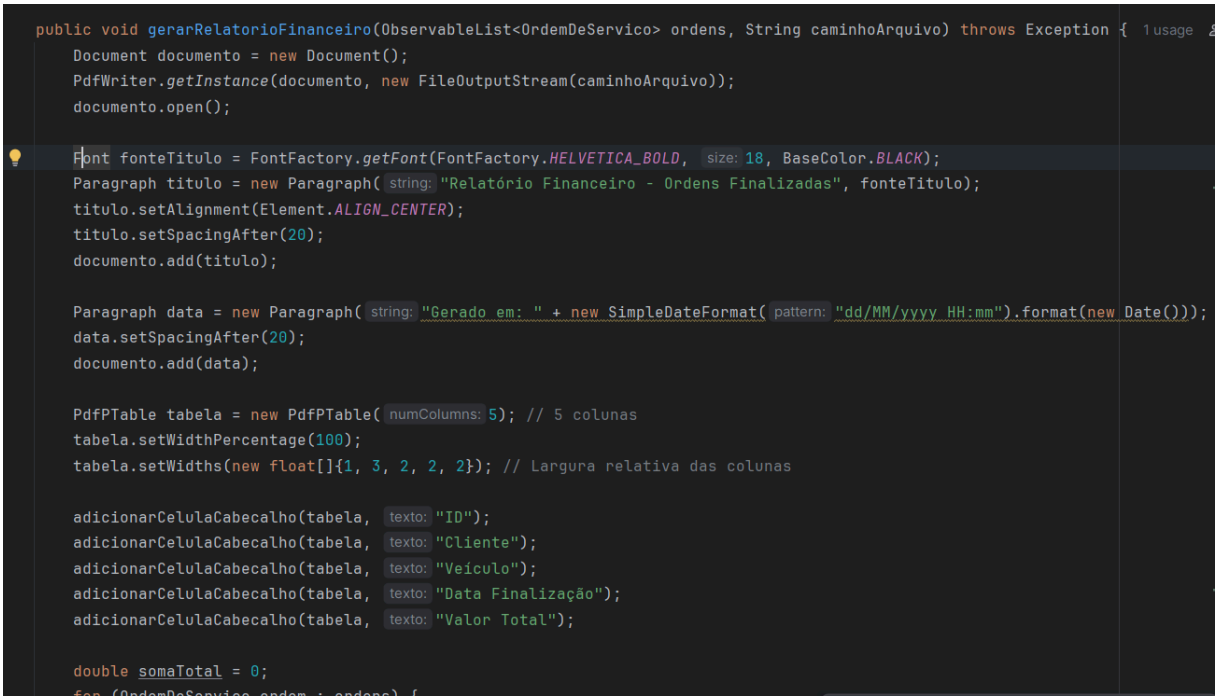
Para o desenvolvimento deste projeto, foram consultadas diversas fontes de documentação técnica e comunidades de desenvolvimento:

- **Documentação Oficial:** Oracle Java Documentation e JavaFX API.
- **Repositórios e Fóruns:**
    - *Maven Repository* (para gestão de dependências como iText e MySQL Connector).
    - *StackOverflow* (para resolução de erros específicos de SQL e NullPointer Exceptions).
    - *Reddit* (comunidades r/javahelp e r/learnprogramming).
- **Tutoriais em Vídeo:**
    - Tutoriais sobre JavaFX e SceneBuilder, especificamente: https://youtu.be/81CK0g4Glw4
- **Ferramentas de IA:** Uso de IA Generativa para auxílio na refatoração de código, otimização de queries SQL e estruturação do padrão MVC.

---

7. Destaque de Código (Classe GeradorPDF)

Captura de Ecrã:



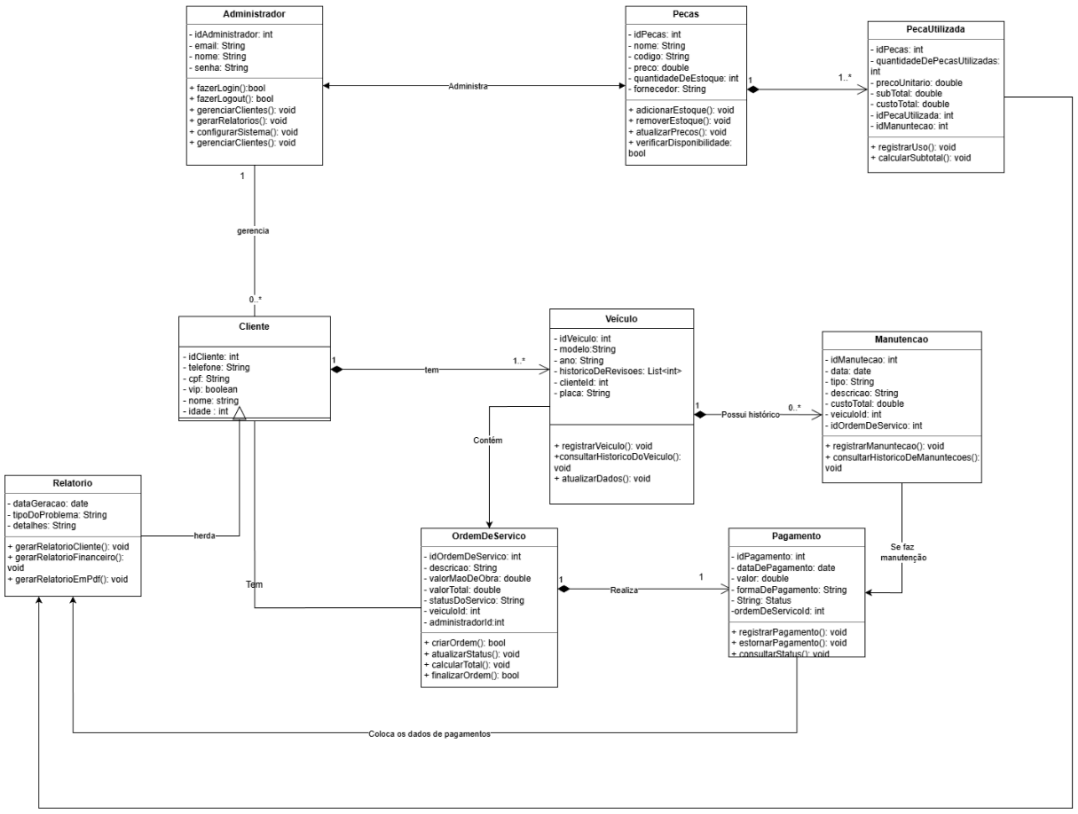
```
public void gerarRelatorioFinanceiro(ObservableList<OrdemDeServico> ordens, String caminhoArquivo) throws Exception {  1 usage
    Document documento = new Document();
    PdfWriter.getInstance(documento, new FileOutputStream(caminhoArquivo));
    documento.open();

    Font fonteTitulo = FontFactory.getFont(FontFactory.HELVETICA_BOLD,  size: 18, BaseColor.BLACK);
    Paragraph titulo = new Paragraph( string: "Relatório Financeiro - Ordens Finalizadas", fonteTitulo);
    titulo.setAlignment(Element.ALIGN_CENTER);
    titulo.setSpacingAfter(20);
    documento.add(titulo);

    Paragraph data = new Paragraph( string: "Gerado em: " + new SimpleDateFormat( pattern: "dd/MM/yyyy HH:mm").format(new Date()));
    data.setSpacingAfter(20);
    documento.add(data);

    PdfPTable tabela = new PdfPTable( numColumns: 5); // 5 colunas
    tabela.setWidthPercentage(100);
    tabela.setWidths(new float[]{1, 3, 2, 2, 2}); // Largura relativa das colunas

    adicionarCelulaCabecalho(tabela,  texto: "ID");
    adicionarCelulaCabecalho(tabela,  texto: "Cliente");
    adicionarCelulaCabecalho(tabela,  texto: "Veículo");
    adicionarCelulaCabecalho(tabela,  texto: "Data Finalização");
    adicionarCelulaCabecalho(tabela,  texto: "Valor Total");

    double somaTotal = 0;
    for (OrdemDeServico ordem : ordens) {
```

Por que razão esta classe é importante?

justificativa:

Essa classe foi trabalhosa, por ser a primeira vez que o grupo mexeu com essa biblioteca, ela é muito importante pois vai gerar o relatório financeiro de todas as ordens de serviços com o ID, nome do cliente, veículo, Data de finalização da ordem e Valor Total daquela ordem, tudo isso no formato de tabela, isso por si só é trabalhoso e precisou de um certo tempo

8. Diagrama UML de casos de uso


Administrador
- idAdministrador: int
- email: String
- nome: String
- senha: String
+ fazerLogin():bool
+ fazerLogout(): bool
+ gerenciarClientes(): void
+ gerarRelatorios(): void
+ configurarSistema(): void
+ gerenciarClientes(): void
Administra
Pecas
- idPecas: int
- nome: String
- codigo: String
- preco: double
- quantidadeDeEstoque: int
- fornecedor: String
+ adicionarEstoque(): void
+ removerEstoque(): void
+ atualizarPrecos(): void
+ verificarDisponibilidade: bool
1
1..*
PecaUtilizada
- idPecas: int
- quantidadeDePecasUtilizadas: int
- precoUnitario: double
- subTotal: double
- custoTotal: double
- idPecaUtilizada: int
- idManuntecao: int
+ registrarUso(): void
+ calcularSubtotal(): void
1
gerencia
0..*
Cliente
- idCliente: int
- telefone: String
- cpf: String
- vip: boolean
- nome: string
- idade : int
1
tem
1..*
Veículo
- idVeiculo: int
- modelo:String
- ano: String
- historicoDeRevisoes: List<int>
- clienteId: int
- placa: String
+ registrarVeiculo(): void
+consultarHistoricoDeVeiculo(): void
+ atualizarDados(): void
1
Possui histórico
0..*
Manutencao
- idManutecao: int
- data: date
- tipo: String
- descricao: String
- custoTotal: double
- veiculoId: int
- idOrdemDeServico: int
+ registrarManuntecao(): void
+ consultarHistoricoDeManuntecoes(): void
Contém
Relatorio
- dataGeracao: date
- tipoDoProblema: String
- detalhes: String
+ gerarRelatorioCliente(): void
+ gerarRelatorioFinanceiro(): void
+ gerarRelatorioEmPdf(): void
herda
Tem
OrdemDeServico
- idOrdemDeServico: int
- descricao: String
- valorMaoDeObra: double
- valorTotal: double
- statusDoServico: String
- veiculoId: int
- administradorId:int
+ criarOrdem(): bool
+ atualizarStatus(): void
+ calcularTotal(): void
+ finalizarOrdem(): bool
1
Realiza
1
Pagamento
- idPagamento: int
- dataDePagamento: date
- valor: double
- formaDePagamento: String
- String: Status
-ordemDeServicoId: int
+ registrarPagamento(): void
+ estornarPagamento(): void
+ consultarStatus(): void
Se faz manutenção
Coloca os dados de pagamentos


Gerenciar cadastro
Includes
Cadastro de cliente
Includes
Cadastro de Veículo
Controle de pagamentos
Administrador
Ver historico de manutenções
Ver Custo de Manutenção
Extends
Extends
Cliente
Gerar relatório
Ver status
Extends
Atualizar status
Criar ordem de serviço
Includes
Priorizar cliente VIP
Gerenciar estoque
Agendar revisão
Estoque de peças
Extends
Histórico de uso