ESCOLA SENAI “HENRIQUE LUPO”
TECNICO EM DESENVOLVIMENTO DE SISTEMAS

André Turquiai Bido
Caio Hiroshi Ferreira
Jessica Pichinin da Costa
Leticia Rodrigues Lemes
Vinicius Amâncio da Silva

MoveBR: movemos a educação que move o mundo!

ARARAQUARA
2024

André Turquiai Bido

Caio Hiroshi Ferreira

Jessica Pichinin da Costa

Leticia Rodrigues Lemes

Vinicius Amâncio da Silva

MoveBR: movemos a educação que move o mundo!

Trabalho de Conclusão de Curso/Projeto apresentado como requisito parcial para a obtenção do certificado de técnico em Desenvolvimento de Sistemas, da Escola SENAI “Henrique Lupo”.

Orientador(a): Prof. Alex Stocco
Prof. Ivo Neto

ARARAQUARA

2024

Aos meus pais,

Aos meus avós,

Ao meu irmão e/ou à minha irmã.

AGRADECIMENTOS

Agradecemos, primeiramente, a Deus, fonte de força e inspiração em toda essa caminhada. Às nossas famílias, que nos apoiaram incondicionalmente, oferecendo amor, paciência e incentivo em cada etapa. Aos colegas e amigos, por estarem ao nosso lado, compartilhando desafios, aprendizados e momentos inesquecíveis. Aos professores Alex Stocco e Ivo Neto, que nos marcaram com sua dedicação, ensinamentos valiosos e exemplos de profissionalismo. Vocês foram essenciais para o nosso crescimento pessoal e acadêmico. A todos que contribuíram para essa conquista, nossa sincera gratidão.

RESUMO

O sistema proposto é uma solução integrada e automatizada para gestão do transporte escolar, projetada para atender às principais demandas dos motoristas de vans escolares. Ele centraliza e otimiza tarefas essenciais, como o controle financeiro, o cadastro detalhado de alunos e a organização das rotinas diárias. A plataforma busca substituir métodos manuais e ferramentas genéricas, oferecendo funcionalidades específicas para o setor, como gerenciamento de pagamentos, registro de informações dos alunos e relatórios simplificados. Além disso, o sistema melhora a eficiência operacional e proporciona mais segurança e transparência para os responsáveis, tornando-se uma ferramenta prática e acessível que facilita a vida dos motoristas e eleva a qualidade do serviço de transporte escolar.

ABSTRACT

The proposed system is an integrated and automated solution for managing school transportation, designed to meet the key needs of school van drivers. It centralizes and optimizes essential tasks, such as financial management, detailed student registration, and the organization of daily routines. The platform replaces manual methods and generic tools by offering features tailored to the sector, such as payment management, student information logging, and simplified reports. Additionally, the system improves operational efficiency and provides greater security and transparency for parents, making it a practical and accessible tool that facilitates drivers' work and enhances the quality of school transportation services.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
API	Application Programming Interface
SQL	Structured Query Language

SUMÁRIO

1. INTRODUÇÃO

O transporte escolar desempenha um papel crucial na garantia do acesso à educação, especialmente em regiões onde a distância entre as residências e as instituições de ensino é significativa. Porém, a complexidade envolvida na gestão eficiente desse serviço apresenta uma série de desafios para os motoristas de vans escolares. O gerenciamento de pagamentos e o controle das informações sobre os estudantes são tarefas que, embora essenciais, são frequentemente realizadas de forma manual ou com o uso de ferramentas genéricas. Essas soluções muitas vezes não atendem às necessidades específicas do setor de transporte escolar, resultando em processos ineficientes, suscetíveis a erros e desgastes operacionais.

A ausência de uma ferramenta especializada agrava os desafios enfrentados pelos motoristas e operadores desse serviço. A comunicação com os pais e responsáveis é um ponto crítico, frequentemente feita de maneira informal e dispersa, o que pode gerar falta de clareza em informações importantes, como horários, mudanças de rota ou questões relacionadas aos pagamentos. Além disso, o controle financeiro, essencial para a manutenção do serviço, também carece de uma gestão eficaz, levando a atrasos, erros nos registros de pagamento e uma falta de visibilidade para os responsáveis sobre a situação financeira.

Desse modo como um sistema automizado pode auxiliar na organização dos motoristas de van? A hipótese do projeto é que a adoção de um sistema automatizado reduzirá significativamente os problemas decorrentes da falta de ferramentas adequadas, aumentando a satisfação tanto dos motoristas quanto dos pais e alunos. O desenvolvimento de um sistema integrado específico para a gestão do transporte escolar torna-se uma solução viável e necessária. O sistema proposto visa automatizar e centralizar todas as tarefas envolvidas no gerenciamento desse serviço, proporcionando uma ferramenta completa que permita aos motoristas controlar os pagamentos de forma eficiente e manter um canal de comunicação claro e ágil com os pais e responsáveis. Acredita-se que, com a implementação dessa solução, os motoristas terão uma redução significativa no trabalho operacional manual, o que resultará em maior eficiência e precisão no dia a dia. Ao automatizar essas tarefas, o sistema também trará mais segurança e confiabilidade para os responsáveis pelos alunos, que poderão acompanhar de perto as atividades e ter

maior transparência sobre a situação financeira e o status do transporte. Além disso, a melhoria na comunicação e na gestão financeira impactará positivamente a confiança e a satisfação dos pais, enquanto os motoristas poderão focar mais em suas responsabilidades principais, minimizando erros e o tempo gasto em processos administrativos.

1.1. Objetivo Geral

Desenvolver um sistema integrado e automatizado de gestão para o transporte escolar, que contemple a organização e otimização das rotas, facilite o controle financeiro dos pagamentos, centralize o cadastro detalhado dos alunos, e melhore a comunicação entre motoristas, responsáveis e alunos.

1.2. Objetivos Específicos

- Realizar entrevistas com motoristas para identificar suas principais demandas e desafios na gestão do transporte escolar.
- Levantar e pesquisar bibliografia sobre práticas de organização de rotas e ferramentas úteis para motoristas de vans escolares.
- Identificar e selecionar as linguagens de programação que serão necessárias para o desenvolvimento do sistema, visando uma interface amigável.
- Desenvolver um protótipo do sistema que facilite a organização das rotas e o gerenciamento diário das atividades do motorista.

1.3. Justificativa

A escolha do desenvolvimento deste sistema de gestão de transporte escolar foi motivada por uma combinação de fatores pessoais e profissionais, que destacam a relevância e a necessidade prática desta solução. O primeiro motivo está relacionado à experiência de um dos membros do grupo, cujo pai é motorista de van escolar. Por meio dessa vivência, ficou claro para nós as dificuldades enfrentadas por esses profissionais, especialmente no que diz respeito à organização de rotas, pagamentos e a comunicação com os responsáveis. Além disso, vários membros do grupo também já utilizaram vans para se deslocar até a escola (SESI), o que nos

permitiu observar a realidade e os desafios desse serviço tanto do ponto de vista dos motoristas quanto dos alunos e seus responsáveis.

A relevância teórica do projeto está na aplicação de tecnologias modernas para resolver um problema cotidiano ainda pouco abordado, contribuindo com uma solução inovadora para um setor tradicional. Metodologicamente, o projeto busca integrar diferentes áreas do desenvolvimento de sistemas, como otimização de rotas, automação financeira e comunicação eficiente, permitindo a aplicação prática dos conhecimentos adquiridos ao longo do curso.

Do ponto de vista social, o sistema proposto pode melhorar significativamente a qualidade do serviço de transporte escolar, trazendo mais segurança e transparência para os pais e responsáveis, além de facilitar o trabalho dos motoristas. A falta de ferramentas específicas para este setor demonstra a necessidade de soluções que contribuam para um serviço mais eficiente e organizado.

Por fim, o desenvolvimento deste sistema também tem uma importância pessoal para os membros do grupo, pois além de ser um desafio técnico relevante, é uma oportunidade de criar algo que impacta diretamente a comunidade e ajuda a resolver problemas reais que eles próprios testemunharam.

2. DESENVOLVIMENTO

2.1. Escolha do tema

O tema do nosso trabalho de conclusão de curso surgiu a partir da observação de uma carência significativa no mercado de transporte escolar. A partir da experiência dos pais de um dos integrantes do grupo, que atuam nesse setor, identificamos que muitos profissionais enfrentam dificuldades na administração de suas atividades diárias, especialmente pela falta de ferramentas específicas para organizar de forma eficiente as informações dos alunos, controlar a parte financeira e gerenciar a logística do serviço. Essa carência de soluções adequadas torna o

trabalho desses profissionais ainda mais complexo e menos eficiente, o que compromete tanto a qualidade do serviço prestado quanto a segurança e o bem-estar dos alunos.

Percebemos que o setor carece de um sistema integrado que atenda de maneira eficaz a essas necessidades, uma vez que, até o momento, não havia uma plataforma específica voltada para o transporte escolar. Com isso, decidimos criar uma solução tecnológica inovadora, que unisse em um único sistema funcionalidades como o cadastro de motoristas, o controle das informações dos alunos e a gestão financeira.

2.2. Entrevistas

Após definirmos o tema do nosso projeto, decidimos aprofundar nossa pesquisa realizando entrevistas com profissionais que atuam diretamente no transporte escolar. O objetivo dessas entrevistas foi entender de forma mais precisa as principais dificuldades e necessidades enfrentadas no dia a dia desses motoristas. A partir dessas informações, buscamos orientar o desenvolvimento de uma solução que realmente resolvesse seus problemas e fosse útil no cotidiano deles.

Durante as entrevistas, fizemos perguntas específicas sobre a organização das rotas diárias, um aspecto fundamental para esses

profissionais. Também procuramos entender que tipo de suporte adicional eles consideram essencial para tornar o trabalho mais eficiente e menos sobrecarregado. Além disso, investigamos como os motoristas já se organizam, quais métodos utilizam atualmente e quais pontos poderiam ser melhorados.

Outro ponto importante foi a comunicação entre motoristas e os responsáveis pelos alunos, geralmente os pais. Perguntamos como é feita a interação, especialmente em relação ao pagamento dos serviços e à rotina escolar dos estudantes. Percebemos que muitos desses processos ainda são realizados de forma manual, sem o auxílio de sistemas adequados, o que causa desorganização e dificuldades operacionais. Essas entrevistas foram essenciais para orientar o desenvolvimento do nosso sistema, permitindo que a ferramenta seja construída com base nas necessidades reais dos profissionais e melhore a gestão das atividades diárias no transporte escolar.

2.3. Requisitos

Após a definição do tema e a realização das entrevistas com profissionais do setor de transporte escolar, buscamos aprofundar nosso entendimento sobre as necessidades reais desses usuários. Com base nas informações coletadas, foi possível identificar as funcionalidades essenciais que a ferramenta deveria ter para atender às demandas de forma eficaz. A partir disso, começamos a elaborar os requisitos do sistema, fundamentais para guiar o seu desenvolvimento e garantir que ele realmente resolvesse os problemas enfrentados pelos motoristas e gestores.

Esses requisitos foram divididos em duas categorias principais: funcionais e não-funcionais, que desempenham papéis complementares na construção de um sistema robusto e eficiente.

2.3.1. Requisitos funcionais

Os requisitos funcionais são essenciais porque definem claramente o que o sistema deve fazer para resolver os problemas dos usuários. Eles têm como objetivo atingir a solução das dificuldades identificadas durante as entrevistas e pesquisas, trabalhando diretamente na entrega das funcionalidades necessárias. Sem esses requisitos, o nosso sistema poderia falhar em oferecer as ferramentas essenciais, como o cadastro de motoristas, o gerenciamento das rotas e o controle de pagamentos. Esses requisitos fornecem uma base sólida para o desenvolvimento, garantindo que a equipe saiba exatamente quais funções precisam ser implementadas e testadas, para que o sistema atenda de forma eficaz às necessidades dos profissionais de transporte escolar.

2.3.2. Requisitos não funcionais

Os requisitos não-funcionais são fundamentais para garantir que o sistema funcione de maneira eficiente e atenda a padrões de qualidade, abordando aspectos como desempenho, segurança, usabilidade e confiabilidade. Esses requisitos são essenciais porque definem como as operações serão realizadas, assegurando que o sistema não só ofereça as funcionalidades certas, mas também opere de forma eficaz, rápida e segura. Por exemplo, garantir que o sistema seja rápido, seguro e fácil de usar é tão importante quanto ter as funcionalidades adequadas. Sem esses requisitos, o sistema poderia apresentar problemas como lentidão, vulnerabilidades de segurança e dificuldades de uso, comprometendo sua eficácia e a aceitação pelos usuários.

Portanto, a integração dos requisitos funcionais e não-funcionais é essencial no desenvolvimento do projeto, pois ela garante que o sistema ofereça as funcionalidades necessárias, ao mesmo tempo em que opera de maneira eficiente, segura e com uma boa experiência de uso. Isso

assegura que o sistema não apenas resolva os problemas do usuário, mas também seja confiável e adequado para o seu uso no dia a dia.

2.3.3. Requisitos funcionais criados para o nosso sistema:

2.3.3.1. RF001 - Cadastro do motorista

O motorista deve se cadastrar no aplicativo, fornecendo informações como nome, CPF, CNPJ, CNH D, cidade, telefone, e-mail, endereço e senha. O cadastro é inserido no banco de dados e, em seguida, o usuário é redirecionado para a página de login. É necessário que o motorista possua a CNH D regularizada, assegurando a habilitação para conduzir veículos com mais de 8 passageiros. Após o login, o motorista acessa a página principal do aplicativo. A prioridade desse requisito é essencial para o funcionamento do serviço.

2.3.3.2. RF002 - Cadastro do aluno

O motorista realiza o cadastro do aluno que irá transportar, assegurando que o aluno esteja matriculado em uma escola. As informações necessárias incluem nome do aluno, nome e telefone do responsável, CPF do responsável, cidade, escola, email, endereço e senha. Após a inserção dos dados no banco de dados, uma mensagem de "Aluno Cadastrado" é exibida. O processo garante que o aluno fique registrado no sistema. Assegura-se a atualização e integridade das informações. A prioridade do cadastro é considerada essencial. Não há dependências associadas.

2.3.3.3. RF003 – Link para cadastro de aluno

O sistema gera um link único para cada aluno, que é enviado aos responsáveis para que eles possam cadastrar o aluno no sistema. O link é enviado automaticamente ao responsável, e a confirmação de envio é registrada. Este requisito depende de outros processos, como o cadastro do motorista e o login.

2.3.3.4. RF004 – Login

O motorista pode fazer login caso já tenha realizado o cadastro. As informações necessárias para o login são o email e a senha. Se as credenciais estiverem corretas, o motorista é direcionado para a página principal do sistema. Isso permite que ele tenha acesso às suas funcionalidades e gerencie o transporte de alunos. O login é considerado essencial para a operação do site. Este processo depende do cumprimento do RF001 (Cadastro do Motorista) e do RF002 (Cadastro do Aluno). Acesso seguro e funcionalidade são prioridades.

2.3.3.5. RF005 - Listar Alunos

O sistema permite que o motorista visualize todos os alunos cadastrados que são seus clientes ao acessar a página de listagem. A pré-condição é que o motorista esteja registrado e que existam alunos já cadastrados no sistema. Não há entradas necessárias para essa funcionalidade. O resultado é uma lista de alunos cadastrados que o motorista pode consultar. A prioridade desse requisito é essencial para que o motorista gerencie seus alunos adequadamente.

2.3.3.6. RF006 - Detalhar aluno

O sistema permite que o motorista veja informações detalhadas sobre um aluno ao clicar na seta do card correspondente. Para acessar essa funcionalidade, é necessário estar na página de listagem de alunos. A entrada consiste na ativação do botão, resultando na exibição das informações adicionais do aluno. Este recurso é essencial para facilitar a gestão do transporte. As dependências incluem RF001 (Cadastro do Motorista), RF002 (Cadastro do Aluno), RF003 (Login) e RF004 (Listar Alunos). A funcionalidade melhora a compreensão das necessidades dos alunos. É uma parte importante da experiência do motorista no sistema.

na tela. A prioridade desse requisito é importante para auxiliar na gestão financeira do motorista.

2.3.3.7. RF007 – Editar aluno

Este requisito permite que o motorista ou administrador do sistema edite os dados dos alunos já cadastrados, como informações pessoais ou de contato. Isso é útil quando há mudanças nas informações dos alunos ou responsáveis. Este requisito depende do cadastro prévio dos alunos e do login do motorista.

2.3.3.8. RF008 – Excluir aluno

Permite que o motorista ou administrador exclua um aluno do sistema. Quando um aluno não é mais transportado, o motorista pode remover o aluno do banco de dados. Esse processo depende do cadastro do aluno e do login do motorista, e é considerado desejável, mas não essencial.

2.3.3.9. RF009 – Criação de rotas

O motorista pode criar rotas específicas para o transporte dos alunos utilizando a API do Google Maps. Ao selecionar os alunos para o dia, o sistema gera a rota mais eficaz para otimizar o transporte. Esse requisito facilita a logística do transporte escolar, sendo desejável, mas não essencial.

2.3.3.10. RF010 – Cadastro de pagamento

O motorista pode registrar os pagamentos realizados pelos alunos, ajudando a controlar a sua renda. O sistema exibe uma tabela onde o motorista seleciona o mês e visualiza os pagamentos feitos ou não feitos pelos alunos. Este requisito é importante para o controle financeiro do motorista.

2.3.3.11. RF011 - Somar pagamentos já feitos

Esta função soma todos os pagamentos efetuados e confirmados durante o mês, exibindo o total ao motorista. A pré-condição é que o motorista esteja na página de histórico de pagamentos. A entrada consiste na ação de acionar o botão para somar. A saída é o total dos pagamentos verificados que foram realizados. A pós-condição é que o resultado final da soma apareça na tela. A prioridade desse requisito é importante para auxiliar na gestão financeira do motorista.

2.3.3.12. RF012 – Editar pagamento

Este requisito permite editar um pagamento que já foi registrado no sistema. Se houver algum erro ou alteração nos detalhes de um pagamento, o usuário autorizado (como o motorista ou administrador) pode corrigir a informação. A prioridade é desejável, mas não essencial.

2.3.3.13. RF013 – Excluir pagamento

Permite que um pagamento registrado seja excluído do sistema, caso necessário. Após confirmação, o pagamento será removido do banco de dados. A exclusão de pagamentos é desejável, mas não essencial, e é útil para corrigir erros no registro de pagamentos.

2.3.3.14. RF014 – Listar Pagamentos Pendentes

O motorista pode visualizar todos os pagamentos que ainda não foram feitos pelos alunos. Esse recurso ajuda a monitorar o status dos pagamentos, facilitando o controle financeiro. A prioridade é desejável e depende do cadastro de pagamentos.

2.3.3.15. RF015 - Quebra de contrato

Esta função permite a exclusão do aluno da tabela de clientes do motorista em caso de quebra de contrato. A pré-condição é que tanto o aluno quanto o motorista estejam cadastrados. A entrada consiste na ação de clicar no botão de cancelamento na página de listagem de alunos. A saída é uma mensagem de confirmação

exibida para o motorista. A pós-condição é que os pagamentos devem estar em dia; caso haja atraso, o aluno deve pagar a multa e o mês não quitado. A prioridade desse requisito é essencial para garantir a gestão adequada dos contratos.

2.3.4. Requisitos não funcionais criados:

2.3.4.1. RNF001 - Desempenho do aplicativo

O requisito não funcional RNF001 garante que o sistema execute rapidamente o cadastro dos motoristas, inserindo os dados fornecidos pelo usuário na tabela "tb_motoristas" de forma eficiente e segura. É essencial para a funcionalidade do sistema, exigindo validação de dados, segurança contra-ataques, e integração com o backend em Python. O desempenho deve ser otimizado para garantir uma boa experiência ao usuário, sem atrasos perceptíveis.

2.4. Regras de negocio

Regras de negócio são instruções específicas que orientam o comportamento de um sistema ou processo dentro de uma organização. Elas definem como certas atividades devem ser conduzidas, com base em políticas, leis, normas e práticas do setor, garantindo que as operações do sistema estejam de acordo com esses parâmetros. As regras de negócio podem incluir validações de dados, restrições operacionais, condições para tomada de decisão e limites para execução de processos.

Essas regras são essenciais porque traduzem as necessidades do negócio em termos técnicos, de modo que os desenvolvedores possam implementá-las no sistema. Elas garantem que o software atenda às expectativas da empresa, respeitando a regulamentação vigente, padrões de qualidade e boas práticas. Em resumo, elas são responsáveis por assegurar que o sistema funcione corretamente, de acordo com os objetivos e a realidade do negócio, proporcionando eficiência, segurança e conformidade.

2.4.1. Regras de negocio criadas:

2.4.1.1. REG001 – Validade da Carteira de Motorista

Essa regra garante que todos os motoristas cadastrados possuam uma carteira de motorista válida e específica para o transporte escolar, conforme exigido por lei. Ao validar as credenciais, o sistema assegura que apenas motoristas qualificados operem as vans, promovendo a segurança e o cumprimento das normas de trânsito.

2.4.1.2. REG002 – Manutenção Preventiva dos Veículos

Esta regra determina que os veículos usados no transporte escolar passem por manutenção preventiva regular. A implementação no sistema pode incluir lembretes de manutenção e registros de inspeção, prevenindo falhas mecânicas e garantindo a segurança dos alunos.

2.4.1.3. REG003 – Controle de Capacidade de Passageiros

Essa regra define que o sistema deve garantir que a capacidade máxima de passageiros de cada van não seja excedida. Isso assegura a segurança dos passageiros, evitando superlotação e proporcionando condições adequadas de transporte. O sistema pode alertar motoristas e gestores quando o limite de passageiros for atingido.

2.5. Wireframe

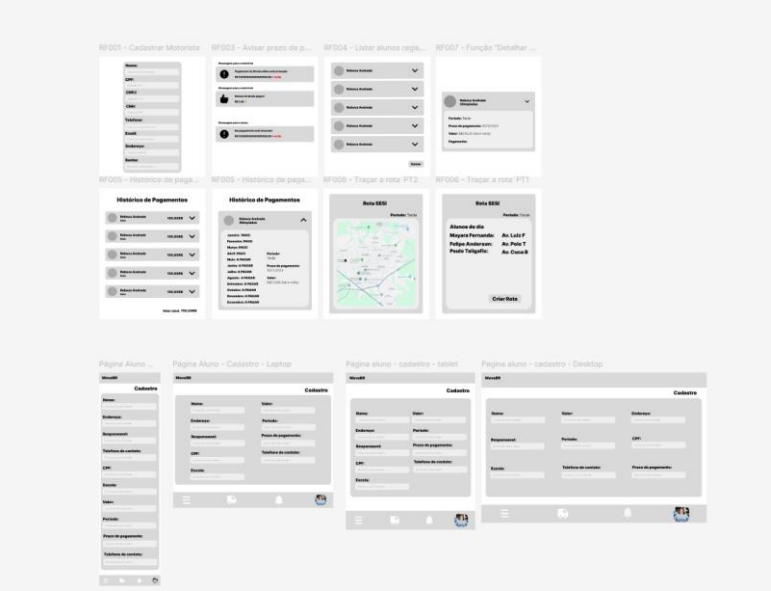
Wireframes são representações visuais simples e esquemáticas de um site ou aplicativo, que destacam a estrutura e a disposição dos elementos na interface do usuário. Eles atuam como um esboço inicial, permitindo que os designers e desenvolvedores visualizem como as páginas e funcionalidades se organizam antes de entrar no design gráfico detalhado.

Esses wireframes são intencionalmente básicos, focando na estrutura e na funcionalidade, em vez de detalhes visuais. Isso ajuda a esclarecer o layout e a interação do usuário. Eles mostram a hierarquia dos elementos na página, como cabeçalhos, menus, botões e campos de formulário, o que é fundamental para a usabilidade e a navegação intuitiva.

A criação de wireframes serve a várias finalidades. Primeiro, eles são uma ferramenta de comunicação entre designers, desenvolvedores e partes interessadas, ajudando todos a estarem alinhados sobre a direção do projeto. Além disso, os wireframes permitem a realização de testes de usabilidade em estágios iniciais, ajudando a identificar problemas de navegação e layout

antes de se investir tempo e recursos no desenvolvimento visual. Eles também funcionam como uma forma de documentação que pode ser referenciada durante o desenvolvimento, assegurando que todos os requisitos e funcionalidades estejam cobertos.

1. Figura 1 Wireframe



Fonte: Autoria própria

2.6. Estudo de cores

2.6.1. O que é

O estudo de cores em um sistema refere-se à análise e escolha estratégica das paletas de cores utilizadas na interface de um aplicativo ou site. Essa prática envolve compreender como diferentes cores interagem entre si, além de considerar a psicologia das cores e seu impacto na percepção do usuário. As cores não apenas ajudam a criar um ambiente visual atraente, mas também desempenham um papel fundamental na comunicação de mensagens e na orientação do usuário. Por exemplo, o uso de cores pode destacar elementos importantes, como botões de ação e informações críticas, além de influenciar o estado emocional dos usuários, criando uma conexão mais profunda com a marca. Em suma, o estudo de cores busca transformar a experiência do usuário em uma jornada intuitiva e agradável, onde a estética e a funcionalidade andam de mãos dadas.

2. Figura 2 – estudo de cores

***IMAGEM**

Fonte: Autoria própria

2.7. Banco de dados

Um banco de dados é uma coleção organizada de informações armazenadas eletronicamente em um sistema de computador, geralmente gerenciado por um sistema de gerenciamento de banco de dados (DBMS). Juntos, os dados e o DBMS formam um sistema de banco de dados.

Os dados são frequentemente estruturados em tabelas com linhas e colunas, o que facilita o acesso, gerenciamento, modificação e consulta. A maioria dos bancos de dados utiliza a linguagem de consulta estruturada (SQL) para operações de escrita e consulta.

2.7.1. SQL

SQL ou Structured Query Language, é uma linguagem de programação fundamental utilizada por praticamente todos os bancos de dados relacionais para consultar, manipular e definir dados, além de fornecer controle de acesso. Desenvolvida pela primeira vez na IBM na década de 1970, com a

colaboração da Oracle, o SQL se tornou a base para o padrão ANSI SQL, que padronizou a forma como interagimos com bancos de dados.

2.7.1.1. Por que usamos SQL?

A linguagem SQL é uma ferramenta amplamente utilizada por sua eficiência na manipulação de dados, permitindo que usuários realizem operações complexas de forma rápida e eficaz. Funções como inserção, atualização e exclusão de dados em grandes volumes de informações podem ser executadas com agilidade, tornando o SQL uma escolha ideal para o gerenciamento de grandes bancos de dados.

Além disso, o SQL facilita a realização de consultas poderosas e intuitivas, onde os usuários podem, com comandos simples, filtrar, ordenar e agregar informações. Isso torna a análise de dados mais acessível e prática, contribuindo para uma gestão de dados eficiente.

Outro ponto forte do SQL é sua padronização globalmente reconhecida. Profissionais de diferentes regiões do mundo podem trabalhar com a mesma linguagem, o que facilita a colaboração e a troca de conhecimentos entre equipes e organizações. Essa uniformidade é um dos fatores que mantêm o SQL relevante no cenário tecnológico.

O SQL também oferece robustos mecanismos de controle de acesso. Administradores podem definir quem tem permissão para visualizar ou manipular dados, garantindo a segurança, integridade e privacidade das informações armazenadas. Isso é fundamental em ambientes onde a proteção de dados é uma prioridade.

2.7.2. Criação do Banco de Dados

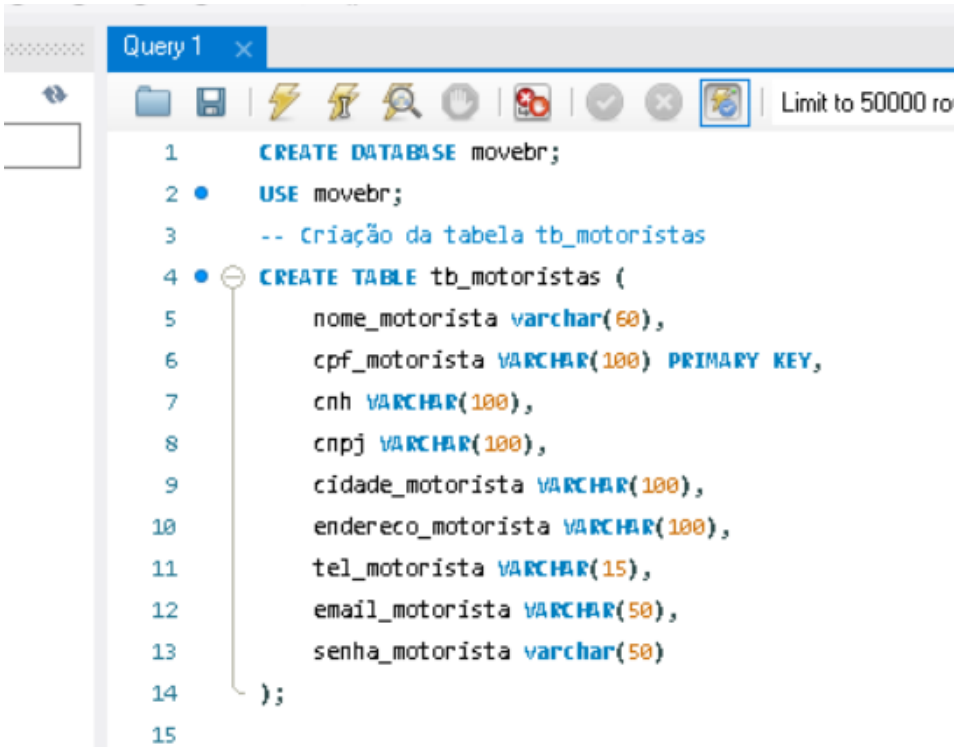
Durante o desenvolvimento do sistema MoveBR, priorizamos a criação de um banco de dados eficiente e estruturado, que fosse capaz de suportar as operações essenciais para o gerenciamento de alunos, motoristas, rotas e pagamentos. A base de dados foi criada com a seguinte estrutura:

2.7.3. Estrutura do Banco de Dados

Inicialmente, criamos o banco de dados com o nome movebr, que será utilizado para armazenar todas as tabelas e informações necessárias ao sistema.

O banco de dados "movebr" foi criado para gerenciar o sistema de transporte escolar de forma eficiente. Ele contém tabelas que armazenam informações relevantes sobre alunos e motoristas, incluindo dados pessoais, registros de pagamentos e faltas. A tabela **tb_alunos** mantém os dados dos estudantes e seus responsáveis, enquanto a **tb_motoristas** armazena as informações dos motoristas de vans escolares. A tabela **tb_alunos_registrados** relaciona alunos a motoristas, facilitando o gerenciamento. O **historico_pagamento** acompanha os pagamentos realizados, e a **tb_faltas** monitora a frequência dos alunos. Além disso, um usuário foi criado para gerenciar o banco, garantindo segurança e controle no acesso às informações. Esse conjunto de tabelas permite uma gestão organizada e eficaz do transporte escolar.

3. Figura 3 – banco de dados



Fonte: Autoria própria

4. Figura 4 – banco de dados

Fonte: Autoria própria

5. Figura 5 – banco de dados

```
-- Criação da tabela historico_pagamento
CREATE TABLE historico_pagamentos (
    nome_aluno VARCHAR(50),
    id_pagamento INT AUTO_INCREMENT PRIMARY KEY,
    valor_pagamento FLOAT,
    mes_pagamento varchar(20),
    data_pagamento varchar(10),
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)

);

-- Criação da tabela verificação_pagamento
CREATE TABLE verificacao_pagamento(
    id_aluno INT,
    valor_pagamento int,
    mes_pagamento int,
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno)
);
```

Fonte: Autoria própria

6. Figura 6 – Banco de dados

```
-- Criação da tabela tb_contratos fechados
CREATE TABLE contratos_fechados (
    id_contrato INT AUTO_INCREMENT PRIMARY KEY,
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);

-- Criação do usuário e concessão de privilégios
CREATE USER 'movebr'@'%' IDENTIFIED BY '123456789';

GRANT ALL PRIVILEGES ON movebr.* TO 'movebr'@'%' WITH GRANT OPTION;
```

Fonte: Autoria própria

2.8. Git Hub

O GitHub é uma plataforma de hospedagem de código-fonte que usa o sistema de controle de versão Git. Ele permite que grupos de desenvolvedores colaborem em projetos, organizando o código, versões e colaborando de forma eficiente.

No contexto do projeto de TCC, o GitHub é utilizado para armazenar o código do sistema, possibilitar que todos os membros do grupo acessem e modifiquem o código de maneira organizada. Aqui está como funciona:

Ao utilizarmos o GitHub no desenvolvimento do nosso projeto de TCC, asseguramos que o código esteja acessível e organizado para todos os membros, facilitando a colaboração e o controle de versões. A plataforma oferece um ambiente seguro, permitindo que cada integrante trabalhe em sua própria branch sem interferir no código principal, garantindo uma integração eficiente das mudanças. Com commits e pull requests, revisamos o código de forma colaborativa, o que promove a qualidade do projeto. Além disso, o sistema de issues nos auxilia na gestão de tarefas e na resolução de problemas, tornando o processo mais ágil e organizado.

2.9. Códigos

Após a fase de planejamento e organização, iniciamos o desenvolvimento do sistema utilizando o Visual Studio Code como ambiente de desenvolvimento integrado (IDE). A implementação foi realizada com uma combinação de tecnologias: HTML e CSS para a construção da interface do usuário, garantindo um layout responsivo e esteticamente agradável. Utilizamos Python para o backend, permitindo a manipulação de dados e a lógica de negócios. JavaScript foi integrado para a interação dinâmica no frontend, proporcionando uma melhor experiência ao usuário. Além disso, APIs foram utilizadas para a comunicação entre diferentes partes do sistema e para integrar funcionalidades externas, resultando em uma aplicação prática e funcional para os motoristas. Essa abordagem modular garante eficiência e escalabilidade ao sistema. Para ficar claro como iremos programar vou explicar cada linguagem.

2.9.1. HTML

HTML, ou HyperText Markup Language, é a linguagem de marcação fundamental para criar e estruturar conteúdo na web. Sua principal função é organizar uma página usando tags específicas, como <h1> para cabeçalhos, <p> para parágrafos e <img> para imagens. HTML também permite a inclusão de links (<a>), facilitando a navegação entre páginas. Essa linguagem é semântica, o que ajuda navegadores e mecanismos de busca a entenderem o significado dos elementos.

A importância do HTML no desenvolvimento web é inegável, pois ele serve como a espinha dorsal de todos os sites e aplicações. Sem HTML, não haveria uma estrutura básica para apresentar informações, tornando impossível a criação de páginas interativas e informativas. Além disso, HTML é frequentemente combinado com CSS (Cascading Style Sheets) para estilização e JavaScript para adicionar interatividade, formando um trio essencial que possibilita a criação de experiências ricas e envolventes. Em resumo, HTML não apenas organiza o conteúdo, mas também permite que desenvolvedores construam uma web dinâmica e acessível.

2.9.2. CSS

CSS, ou Cascading Style Sheets, é uma linguagem de estilo que define a apresentação e o layout de documentos HTML, abrangendo aspectos como cores, fontes, tamanhos e espaçamentos. Sua importância no desenvolvimento de sistemas se destaca na melhoria da aparência visual das interfaces, aumentando o engajamento dos usuários e tornando a navegação mais intuitiva.

Além de proporcionar designs atraentes, o CSS permite a criação de layouts responsivos que se adaptam a diferentes dispositivos, o que é essencial na era digital atual. A separação entre conteúdo e apresentação facilita a manutenção do código, permitindo alterações de estilo sem impactar a estrutura do conteúdo. Com estilos aplicados de maneira consistente, é possível garantir uniformidade em todo o sistema, enquanto animações e transições enriquecem a interatividade e o dinamismo das aplicações web.

2.9.3. JavaScript

JavaScript é uma linguagem de programação de alto nível, amplamente utilizada para adicionar interatividade e dinamismo a páginas web. Como uma linguagem interpretada, JavaScript é executado no lado do cliente, permitindo que os desenvolvedores criem experiências de usuário envolventes, como animações, validação de formulários e atualizações de conteúdo em tempo real, sem a necessidade de recarregar a página. Além disso, JavaScript suporta a programação orientada a objetos e funcional, proporcionando flexibilidade na forma como os desenvolvedores abordam a solução de problemas.

A importância do JavaScript no desenvolvimento web moderno não pode ser subestimada. Ele é um dos pilares fundamentais das tecnologias front-end, frequentemente utilizado em conjunto com HTML e CSS para criar interfaces ricas e interativas. Com o advento de frameworks e bibliotecas como React, Angular e Vue.js, JavaScript evoluiu para suportar aplicações web complexas e escaláveis, permitindo o desenvolvimento de aplicações de página única (SPAs) de

forma eficiente. Além disso, com o Node.js, JavaScript também é amplamente utilizado no lado do servidor, permitindo que os desenvolvedores usem uma única linguagem em todo o stack de desenvolvimento, o que melhora a produtividade e a coesão entre equipes de desenvolvimento.

2.9.4. Python

Python é uma linguagem de programação de alto nível, amplamente utilizada por sua simplicidade e legibilidade. Ela é conhecida por sua sintaxe clara, que permite aos desenvolvedores expressar conceitos de forma concisa e intuitiva. Python suporta múltiplos paradigmas de programação, incluindo programação procedural, orientada a objetos e funcional, o que a torna flexível para diversas aplicações. A linguagem também possui uma vasta biblioteca padrão e uma rica coleção de pacotes de terceiros, facilitando o desenvolvimento de aplicações em áreas como web, ciência de dados, inteligência artificial, automação e muito mais.

A importância do Python no desenvolvimento de sistemas é significativa devido à sua versatilidade e eficiência. Ele é frequentemente escolhido por iniciantes e profissionais, pois permite o desenvolvimento rápido e eficiente de soluções, reduzindo o tempo de codificação. Além disso, Python tem uma comunidade ativa que contribui com recursos, suporte e bibliotecas, o que acelera o aprendizado e a resolução de problemas. Sua integração com outras tecnologias e plataformas torna o Python uma escolha popular para desenvolvimento de back-end, scripts, análise de dados e machine learning, solidificando seu papel como uma das principais linguagens de programação do mundo.

2.9.5. APIs

APIs, ou Application Programming Interfaces, são conjuntos de definições e protocolos que permitem que diferentes sistemas se comuniquem entre si. Elas definem métodos e formatos de requisição e resposta, permitindo que desenvolvedores acessem funcionalidades ou

dados de um serviço externo de forma padronizada. As APIs são fundamentais para a integração de aplicativos e serviços, facilitando a troca de informações entre diferentes plataformas e sistemas.

A importância das APIs no desenvolvimento de software é significativa, pois elas permitem a criação de aplicações mais ricas e interconectadas. Com o uso de APIs, desenvolvedores podem incorporar funcionalidades de terceiros, como sistemas de pagamento, serviços de geolocalização e redes sociais, sem precisar reinventar a roda. Isso acelera o desenvolvimento e reduz custos, uma vez que permite reutilizar serviços já existentes. Além disso, as APIs promovem a modularidade e a escalabilidade das aplicações, permitindo que partes do sistema sejam atualizadas ou substituídas independentemente. Com a crescente adoção de arquiteturas baseadas em microserviços, as APIs se tornaram ainda mais essenciais, pois facilitam a comunicação e a colaboração entre serviços de forma eficiente e organizada.

2.9.5.1. Frontend e Backend

No desenvolvimento de sistemas, o frontend e o backend desempenham papéis complementares. O frontend é a parte visível e interativa de uma aplicação, responsável por tudo o que o usuário vê e interage diretamente. Tecnologias como HTML, CSS e JavaScript são fundamentais aqui. O HTML organiza a estrutura do conteúdo da página, o CSS estiliza a interface, tornando-a visualmente atraente, e o JavaScript adiciona interatividade, como animações, validação de formulários e atualizações dinâmicas sem recarregar a página.

Por outro lado, o backend é o cérebro da aplicação, processando as lógicas de negócios, gerenciando dados e fazendo a comunicação com bancos de dados e APIs. O Python é uma das linguagens mais utilizadas para o desenvolvimento

backend, permitindo manipular dados e automatizar processos do servidor. Além disso, as APIs (Application Programming Interfaces) são fundamentais para conectar o frontend e o backend, permitindo que o frontend faça requisições de dados ao backend de forma padronizada e receba respostas, seja para buscar informações de um banco de dados, autenticar usuários ou realizar outras operações essenciais.

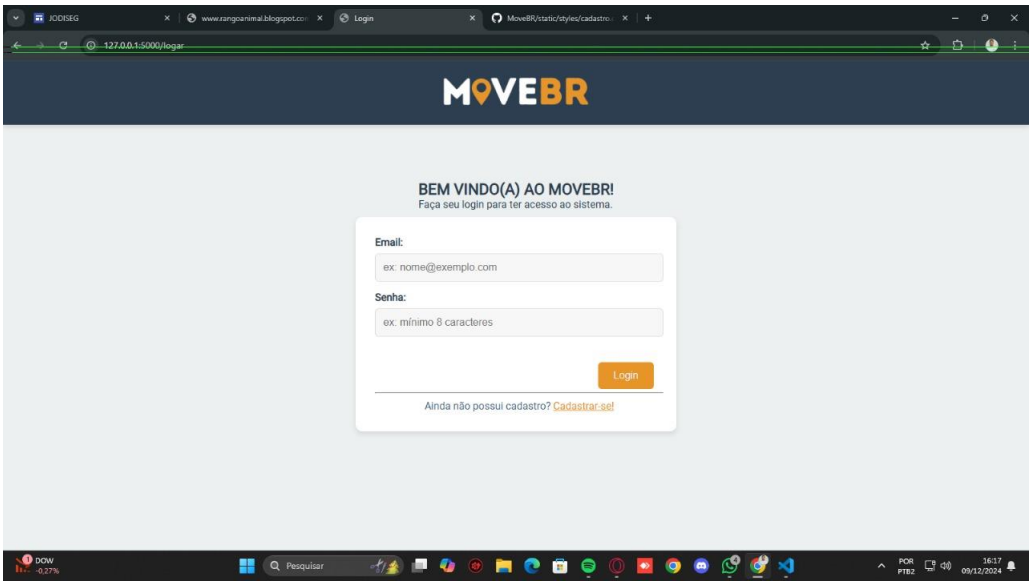
Essa interação entre frontend e backend garante que, enquanto o usuário navega pela interface (frontend), todas as operações complexas e de processamento (backend) sejam executadas em segundo plano, sem sobrecarregar a experiência do usuário.

3. Paginas

3.1. Login

O sistema de gestão de transporte escolar permite que motoristas acessem suas funcionalidades através de uma tela de login, onde devem inserir seu e-mail e senha. Para motoristas que ainda não possuem uma conta, há a opção de se cadastrar, preenchendo um formulário com informações pessoais e criando uma senha. Após o cadastro, um e-mail de confirmação será enviado.

A segurança é uma prioridade, com protocolos de criptografia e opções para recuperação de senha, garantindo que apenas motoristas registrados possam acessar o sistema. Essa estrutura assegura a privacidade das informações, permitindo que os motoristas gerenciem suas atividades de forma eficiente e segura.

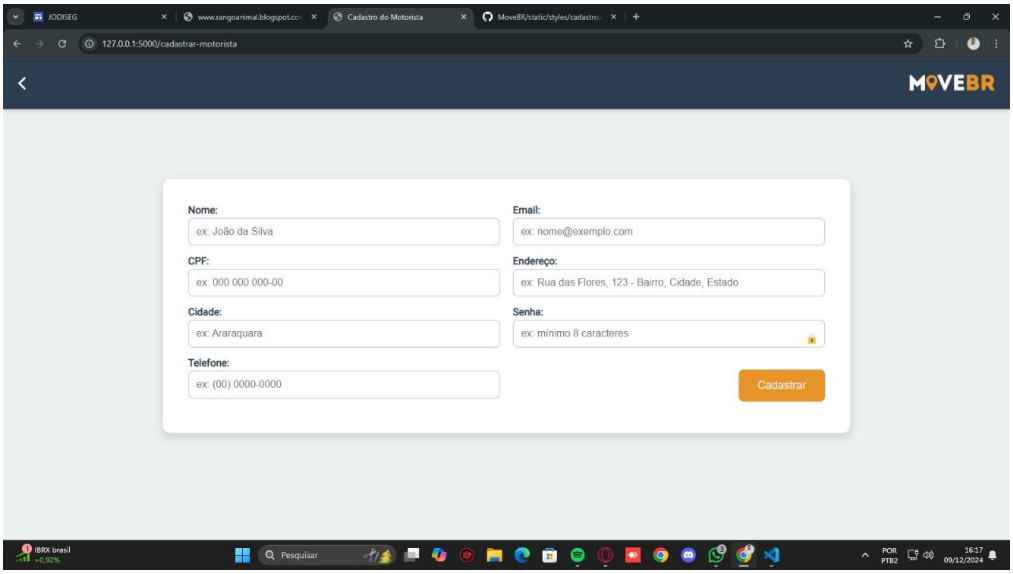


3.2. Cadastro do Motorista

O cadastro do motorista no sistema de gestão de transporte escolar é simples e direto. Na tela de registro, o motorista deve preencher um formulário com informações essenciais, como nome completo, e-mail, telefone e uma senha. Após completar todos os campos obrigatórios, o motorista deve aceitar os termos de uso e enviar o formulário.

Após o envio, o sistema processa o cadastro e envia um e-mail de confirmação para o endereço informado. Isso garante que a conta foi criada com sucesso. O motorista, então, poderá acessar o sistema,

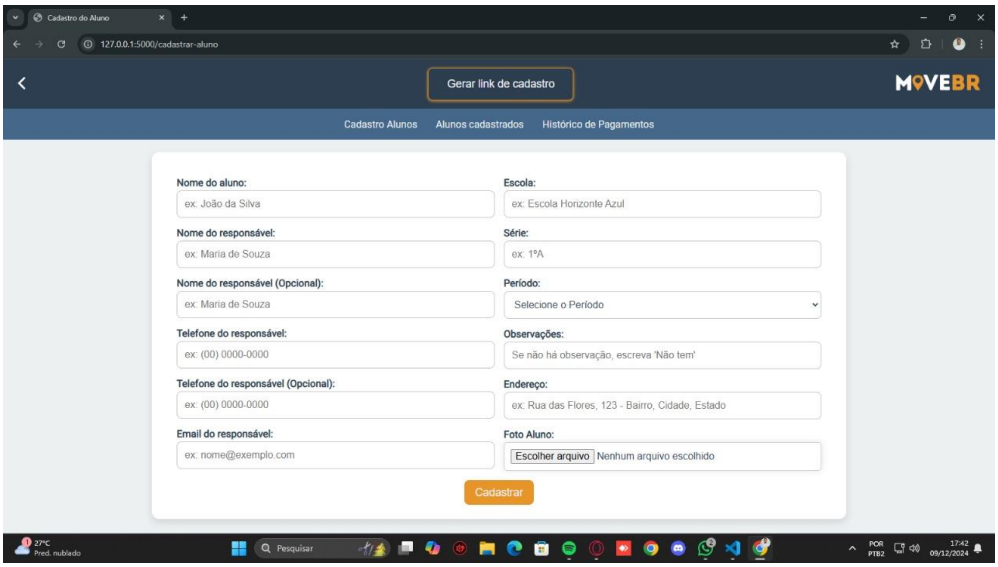
usufruindo de suas funcionalidades para otimizar suas rotas e gerenciar pagamentos, sempre com a segurança de que suas informações estão protegidas.



3.3. Pagina Inicial

A página inicial do sistema de gestão de transporte escolar oferece uma interface intuitiva, com botões que facilitam a navegação. O motorista encontrará opções claras para cadastrar novos alunos, acessar a listagem de alunos já registrados e visualizar o histórico de pagamentos. Essa estrutura organizada permite que o motorista realize suas tarefas de forma eficiente.

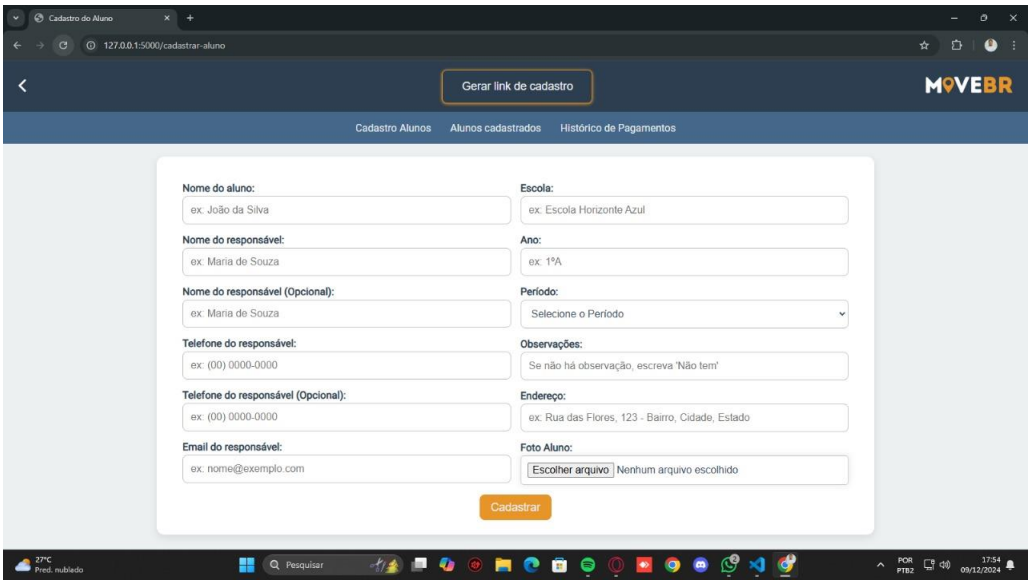
Cada botão direciona o usuário à funcionalidade correspondente, simplificando o processo de gerenciamento. O cadastro de alunos permite a inclusão rápida de informações, enquanto a listagem oferece uma visão geral dos estudantes sob sua responsabilidade. Além disso, o histórico de pagamentos ajuda o motorista a monitorar recebimentos e garantir uma gestão financeira eficaz, tudo em um ambiente seguro e acessível.



3.4. Cadastro Aluno

O cadastro do aluno no sistema de gestão de transporte escolar é um processo fácil e rápido. O motorista acessa a seção de cadastro, onde encontrará um formulário a ser preenchido com informações essenciais, como nome completo do aluno, data de nascimento, endereço, e o nome dos responsáveis. Esses dados são fundamentais para garantir uma gestão adequada e a segurança durante o transporte.

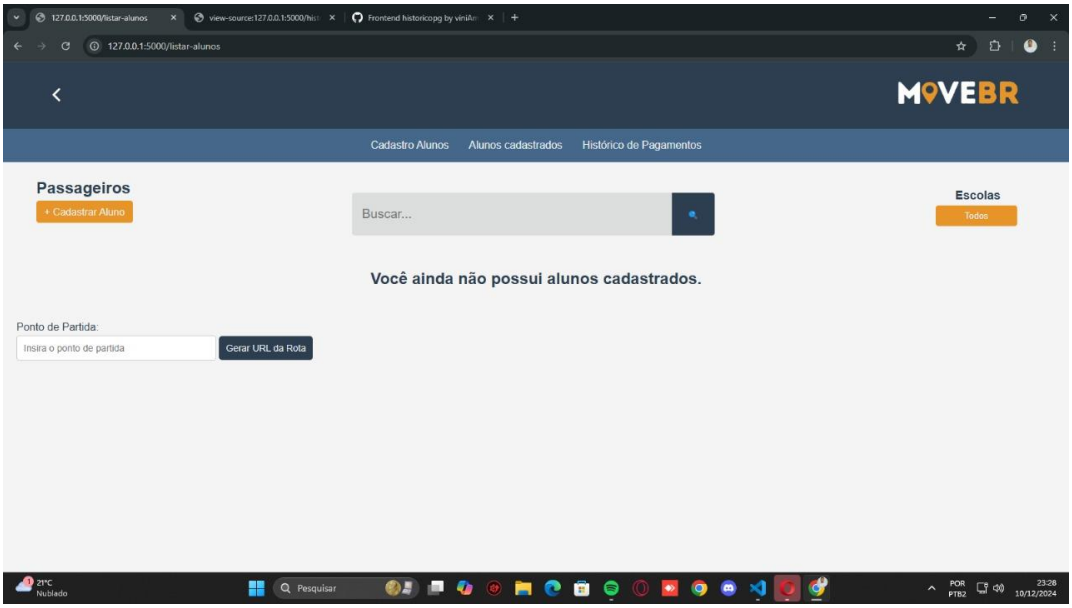
Após preencher o formulário, o motorista submete as informações, que são então armazenadas no sistema. Um e-mail de confirmação pode ser enviado aos responsáveis para garantir que as informações estão corretas. Com o aluno cadastrado, o motorista poderá gerenciar melhor as rotas e acompanhar detalhes importantes, promovendo um serviço mais organizado e eficiente.



3.5. Listar Aluno

A página de listagem de alunos apresenta uma visão clara e organizada de todos os estudantes cadastrados no sistema. O motorista poderá visualizar informações essenciais, como nome, idade, endereço e status de pagamento, tudo em uma tabela de fácil leitura. Essa estrutura permite que o motorista identifique rapidamente cada aluno e suas respectivas necessidades.

Além disso, a página oferece funcionalidades adicionais, como filtros e opções de busca, que facilitam a localização de alunos específicos. O motorista pode também editar ou remover registros, garantindo que as informações estejam sempre atualizadas. Com essa interface, a gestão de alunos se torna mais eficiente, permitindo ao motorista focar em suas responsabilidades principais.



3.6. Detalhar Aluno

Na página de listagem de alunos, ao clicar em um registro específico, o motorista é direcionado para uma página de detalhamento. Nela, são exibidas informações completas sobre o aluno, como dados pessoais, histórico de pagamentos, rotas associadas e contatos dos responsáveis. Essa visão detalhada permite ao motorista entender melhor a situação de cada estudante e suas necessidades específicas.

Além disso, o detalhamento oferece opções para adicionar notas ou anotações sobre o aluno, como instruções especiais ou alertas relevantes. O motorista também pode atualizar informações, como mudanças de endereço ou status de pagamento, garantindo que todos os dados estejam sempre atualizados. Essa funcionalidade contribui para uma gestão mais eficaz e personalizada do transporte escolar.

3.7. Editar perfil

A página "Editar Perfil" permite ao usuário (motorista ou aluno) alterar suas informações pessoais, como nome, e-mail, telefone e outros dados. Ela oferece campos editáveis para que o usuário possa atualizar suas informações, garantindo que os dados no sistema estejam sempre corretos. Após a atualização, o usuário pode salvar as alterações, e o sistema confirmará a modificação. Essa página tem como objetivo proporcionar autonomia ao usuário para gerenciar suas informações de forma simples e eficaz.

3.8. Quebra de contato

A página "Quebra de Contrato" permite ao motorista ou administrador encerrar o serviço de transporte para um aluno. O usuário seleciona o aluno e clica em "cancelar contrato", o sistema verifica se há pendências financeiras e, caso necessário, aplica multas ou cobrança de valores não pagos antes de finalizar a ação. Após a confirmação, o aluno é removido da lista de clientes do motorista.

3.9. Gerar rotas

A página "Gerar Rotas" utiliza a API do Google Maps para criar rotas otimizadas com base nos alunos selecionados pelo motorista. O motorista escolhe os alunos a serem transportados no dia e o sistema gera uma rota eficiente, considerando os endereços dos alunos e a melhor forma de transporte, economizando tempo e combustível.

3.10. Historico de Pagamento

A página de histórico de pagamento fornece ao motorista uma visão detalhada de todas as transações financeiras relacionadas aos alunos. Nela, são listados os pagamentos realizados, incluindo datas, valores e status de cada transação, facilitando o acompanhamento das receitas. Essa organização permite ao motorista identificar rapidamente quais pagamentos estão pendentes ou em atraso.

Além disso, a página oferece funcionalidades para filtrar ou buscar pagamentos por aluno, período ou status. O motorista pode também

adicionar observações ou notas a cada transação, garantindo um registro claro e completo. Com essas informações, o motorista consegue gerenciar melhor sua situação financeira, promovendo um serviço mais transparente e eficiente.

3.11. Gerar pagamentos

A funcionalidade de geração e soma de pagamentos permite ao motorista visualizar rapidamente o total de receitas acumuladas em um determinado período. Na página de histórico de pagamento, há um botão que, ao ser clicado, calcula automaticamente o total de pagamentos recebidos, destacando também os valores pendentes e em atraso. Essa visão clara facilita a gestão financeira e a tomada de decisões.

Além disso, o sistema pode gerar relatórios resumidos que detalham a performance financeira ao longo do tempo, permitindo que o motorista identifique tendências e padrões nos pagamentos. Com essa ferramenta, o motorista tem uma visão abrangente de sua situação financeira, ajudando a planejar melhor as operações do transporte escolar e a garantir a sustentabilidade do serviço.

3.12. Editar Pagamento

A página "Editar Pagamento" permite que o motorista ou administrador altere informações relacionadas a um pagamento registrado. Caso haja algum erro no valor, data ou outro dado de pagamento, o usuário autorizado pode editar essas informações diretamente na página. O sistema exibe um formulário com os detalhes do pagamento, e o usuário pode modificar os campos necessários. Após as alterações, o sistema salva as edições e confirma que o pagamento foi atualizado. Isso garante que o controle financeiro seja mantido de forma precisa, permitindo ajustes sempre que necessário.

3.13. Quebra de contato

O botão de quebra de contrato permite que o motorista registre a solicitação de cancelamento do serviço por parte dos responsáveis. Ao clicar nesse botão, o motorista é direcionado para uma página onde deve confirmar a intenção de encerrar o contrato e preencher informações adicionais, como a data do cancelamento e o motivo. Essa funcionalidade assegura que o processo seja documentado e organizado.

Após preencher as informações necessárias, o motorista confirma a quebra do contrato. O sistema então atualiza o status do aluno e notifica os responsáveis sobre a finalização do serviço. Essa transparência no processo é fundamental, garantindo que todas as partes envolvidas estejam cientes da mudança e mantendo um registro claro para futuras referências.

3 CONSIDERAÇÕES FINAIS

O desenvolvimento deste sistema de gestão para transporte escolar demonstrou como uma ferramenta especializada pode simplificar o trabalho diário dos motoristas e organizar melhor o serviço. Durante o projeto, focamos em criar uma solução que centralizasse as informações dos alunos e tornasse mais fácil o controle financeiro, eliminando a necessidade de processos manuais que tornam a rotina mais desgastante e sujeita a erros.

Os resultados obtidos mostram que a automação realmente torna o gerenciamento mais ágil e preciso, beneficiando diretamente os motoristas, que passam a ter mais organização e controle sobre suas atividades. O sistema proposto é uma solução prática e eficaz, melhorando a operação de transporte escolar e oferecendo uma estrutura que facilita a gestão financeira e o acompanhamento das informações dos alunos. Dessa forma, alcançamos o objetivo de proporcionar uma gestão mais eficiente e simplificada, refletindo na qualidade do serviço e no dia a dia dos motoristas.

REFERÊNCIAS

Icons8: *Icons8*. Disponível em: https://icons8.com.

Google Fonts: *Google Fonts*. Disponível em: https://fonts.google.com.

Bootstrap: *Bootstrap*. Disponível em: https://getbootstrap.com

W3Schools: *W3Schools*. Disponível em: https://www.w3schools.com

GLOSSÁRIO

- API: Interface de Programação de Aplicações, um conjunto de rotinas que permite a comunicação entre diferentes sistemas ou componentes de software.

- Automatização: Processo de implementar sistemas que realizam tarefas manualmente, visando aumentar a eficiência e reduzir erros.

- Banco de Dados: Sistema organizado para armazenar, gerenciar e recuperar informações, permitindo a persistência de dados.

- Back-end: Parte do desenvolvimento que lida com a lógica de negócios, servidor e banco de dados, responsável pelo processamento e armazenamento de dados.

- Branch: Uma cópia separada de um projeto em um repositório do Git, onde desenvolvedores podem trabalhar em alterações sem afetar o código principal.

- CSS: Linguagem de Estilização em Cascata, utilizada para definir a apresentação visual de documentos HTML.

- Commit: Ação de salvar alterações no repositório de um projeto, registrando uma versão específica do código.

- Flask: Microframework para desenvolvimento de aplicações web em Python, que facilita a criação de APIs e serviços web.

- Front-end: Parte do desenvolvimento que se concentra na interface do usuário, envolvendo a criação de layouts e interações visíveis em um navegador.

- Funcionalidades: Conjunto de recursos e ações que o sistema oferece aos usuários, como cadastro de motoristas e gerenciamento de rotas.

- Git: Sistema de controle de versão que permite o gerenciamento de alterações em arquivos e projetos de software.

- Issue: Um item criado no GitHub para rastrear bugs, melhorias ou tarefas a serem realizadas em um projeto.

- HTML: Linguagem de Marcação de Hipertexto, utilizada para estruturar conteúdo na web, formando a base de páginas da internet.

- Interfaces: Conjuntos de elementos visuais que permitem a interação do usuário com um sistema, facilitando a comunicação entre o usuário e a aplicação.

- Merge: O processo de integrar alterações feitas em uma branch ao código principal de um repositório Git.

- Metodologia: Conjunto de métodos e técnicas utilizadas no desenvolvimento do projeto, abrangendo a aplicação prática dos conhecimentos teóricos adquiridos.

- Pull Request: Solicitação para mesclar alterações feitas em uma ramificação de um repositório para outra, geralmente utilizada em projetos colaborativos.

- Protótipo: Representação interativa do site ou aplicativo, que simula a funcionalidade e o design final, permitindo que stakeholders testem e interajam com a interface.

- Quantidade de: Refere-se à forma como as informações são armazenadas e manipuladas dentro de um banco de dados.

- Repositório: Uma pasta online no GitHub onde o código-fonte e outros arquivos de um projeto são armazenados e organizados.

- Responsivo: Um design que se adapta a diferentes tamanhos e formatos de tela, proporcionando uma experiência de usuário consistente em dispositivos variados.

- SQL: Linguagem de Consulta Estruturada, utilizada para gerenciar e manipular bancos de dados relacionais.

- Stakeholders: Pessoas ou grupos que têm interesse ou são afetados pelo projeto, incluindo usuários finais, desenvolvedores e investidores.

- Usabilidade: Facilidade com que os usuários podem interagir com o sistema, influenciando diretamente sua satisfação e eficiência ao utilizar a aplicação.

- Wireframe: Representação visual simples da estrutura e disposição de um aplicativo ou site, focando na funcionalidade e interação do usuário, sem detalhes gráficos.

APÊNDICE A: Login

1-HTML

O código HTML é responsável por definir a estrutura de um formulário de login, que será utilizado para autenticação dos motoristas no sistema. O formulário utiliza o método POST, enviando as informações de login diretamente para o servidor. Ele é dividido em três seções principais. A primeira seção, denominada cima-container, contém os campos de entrada de e-mail e senha, ambos devidamente configurados com seus atributos de type específicos para garantir a validação correta e uma experiência mais segura ao usuário. Além disso, há um link para recuperação de senha em caso de esquecimento, reforçando a usabilidade do sistema. A segunda seção, chamada btn-container, é dedicada ao botão de submissão, que, quando clicado, envia os dados para serem processados. A última parte, baixo-container, apresenta um separador visual (linha horizontal) e um convite para novos motoristas se cadastrarem no sistema, com um link direto para a página de cadastro.

```
 <main>
        <section class="titulo">
            <h2>BEM VINDO(A) AO MOVEBR!</h2>
            <p>Faça seu login para ter acesso ao sistema.</p>
        </section>

        <form class="container-form" action="#" method="post">
            <div class="itens-form">
                <div class="cima-container">
                    <label for="emailAluno">Email:</label>
                    <input type="email" id="emailAluno" name="email"
placeholder="ex: nome@exemplo.com" required>

                    <label for="senhaAluno">Senha:</label>
```

```
                    <input type="password" id="senhaAluno" name="senha"
placeholder="ex: mínimo 8 caracteres" required>
                    <a href="#" class="link-senha">Esqueceu sua
senha?</a>
                </div>

                <div class="btn-container">
                    <button type="submit" class="login">Login</button>
                </div>

                <div class="baixo-container">
                    <hr>
                    <p>Ainda não possui cadastro? <a href="/cadastrar-
motorista">Cadastrar-se!</a></p>
                </div>
            </div>
        </form>
    </main>
```

2-CSS

O código CSS é utilizado para estilizar o formulário de login, proporcionando um design consistente e agradável ao usuário, além de garantir a responsividade e a acessibilidade do layout. Ele começa com um reset básico para eliminar as discrepâncias de estilo entre diferentes navegadores. O cabeçalho é estilizado com uma cor azul escuro e texto claro, criando um efeito visual elegante. O elemento main é centrado usando flexbox, garantindo que o formulário permaneça alinhado verticalmente na tela, independente do tamanho do dispositivo. O formulário, .container-form, tem uma largura fixa, mas responsiva, e utiliza bordas arredondadas para dar um aspecto moderno. Os campos de entrada de e-mail e senha possuem padding adequado, bordas leves e fundo claro, otimizando a usabilidade. Os botões de submissão incluem transições CSS que proporcionam efeitos de hover e active, melhorando a interatividade. Por fim, o rodapé mantém o design unificado e é fixado na parte inferior da página, com a utilização de cores e estilos coerentes.

```
@import url(https://fonts.googleapis.com/css?family=Roboto);

/* Reset básico */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
```

```
}

/* Estilização do corpo */
body {
    font-family: 'Roboto', sans-serif;
    line-height: 1.6;
    background-color: #ecf0f1; /* Cor de fundo clara */
    color: #2c3e50; /* Cor do texto */
}

/* Estilização do cabeçalho */
header {
    background-color: #2c3e50; /* Azul escuro */
    color: #ecf0f1; /* Texto claro */
    height: 80px;
    display: flex;
    justify-content: center;
    align-items: center;
}

.cabecalho_motorista .logo img {
    height: 100px; /* Tamanho da imagem do logo */
}

/* Estilização do main */
main {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    min-height: calc(100vh - 160px); /* Altura mínima */
    padding: 20px;
}

.titulo{
    margin-bottom: 10px;
}

/* Estilização do container-form */
.container-form {
    width: 500px;
    padding: 30px;
    background-color: #ffffff; /* Fundo branco */
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Sombra */
}
```

```
/* Layout dos itens do formulário */
.itens-form {
    display: block;
    flex-wrap: wrap;
    gap: 20px; /* Espaçamento entre os containers */
}

/* Estilização dos containers de lado */
.cima-container,
.baixo-container {
    flex: 1;
    min-width: 450px; /* Garantir largura mínima */
}

.cima-container {
    margin-bottom: 20px;
}

.baixo-container p {
    color: #34495e; /* Azul mais claro */
    text-align: center;
    margin-top: 10px;
}

.baixo-container a {
    color: #E79428; /* Cor vibrante para links */
}

/* Estilização dos labels */
.container-form label {
    display: block;
    margin-bottom: 8px;
    font-weight: 600; /* Peso de fonte mais forte */
    color: #2c3e50; /* Azul escuro para os labels */
}

/* Estilização dos campos de entrada */
.container-form input[type="email"],
.container-form input[type="password"] {
    width: 100%;
    padding: 12px;
    margin-bottom: 15px;
    border: 1px solid #ddd; /* Borda leve */
    border-radius: 6px; /* Bordas arredondadas */
    font-size: 16px;
```

```
    background-color: #f9f9f9; /* Fundo claro */
}

/* Estilização do link de senha */
.link-senha {
    color: #34495e; /* Cor do link */
    font-size: 13px;
}

/* Estilização do botão */
.btn-container {
    width: 100%;
    display: flex;
    justify-content: flex-end;
}

.btn-container button {
    background-color: #E79428; /* Cor do botão */
    color: #ecf0f1; /* Texto claro */
    border: none;
    padding: 12px 24px;
    border-radius: 6px; /* Bordas arredondadas */
    cursor: pointer;
    font-size: 16px;
    margin: 5px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.btn-container button:hover {
    background-color: #d6821e; /* Cor ao passar o mouse */
}

.btn-container button:active {
    transform: scale(0.98); /* Efeito de clique */
}

/* Media Queries */

/* Para telas menores que 1200px */
@media (max-width: 1200px) {
    .container-form {
        width: 90%; /* Largura responsiva */
    }
}

/* Para telas menores que 992px */
```

```
@media (max-width: 992px) {
    header {
        flex-direction: column; /* Coloca a logo e o menu em coluna */
        align-items: center; /* Centraliza a logo */
        text-align: center; /* Centraliza o texto do cabeçalho */
    }

    .cabecalho_motorista {
        padding: 10px 0; /* Espaçamento para melhor visualização */
    }

    .titulo h2 {
        font-size: 22px; /* Ajuste do título */
    }

    .titulo p {
        font-size: 14px; /* Ajuste do texto */
    }
}

/* Para telas menores que 768px */
@media (max-width: 768px) {
    .cima-container,
    .baixo-container {
        flex: 1;
        min-width: 100%; /* Garantir que os containers não fiquem muito
estreitos */
    }

    .container-form {
        width: 100%; /* Largura total */
    }
}

/* Para telas menores que 576px */
@media (max-width: 576px) {
    .container-form {
        width: 100%; /* Largura total */
        padding: 20px; /* Reduzir padding */
    }

    .titulo h2 {
        font-size: 18px; /* Ajuste do título */
    }

    .titulo p {
```

```
        font-size: 12px; /* Ajuste do texto */
    }

    .btn-container {
        flex-direction: column; /* Coloca os botões em coluna */
    }

    .btn-container button {
        width: 100%; /* Botão ocupa toda a largura */
        margin: 5px 0; /* Ajustar margens entre os botões */
    }

    .link-senha {
        font-size: 12px; /* Ajuste do link de senha */
    }
}
```

3-Bancos de dados

A tabela tb_motoristas no banco de dados armazena as informações dos motoristas cadastrados no sistema, como o nome, CPF (que funciona como chave primária), CNH, CNPJ, cidade, endereço, telefone, e-mail e senha. Além disso, são armazenadas imagens da van e do motorista, bem como o valor cobrado pela mensalidade. Essa estrutura organizada facilita a gestão dos dados, permitindo consultas eficientes e comunicação simplificada entre motoristas e a administração, além de garantir a segurança e integridade das informações armazenadas.

```
-- Criação da tabela tb_motoristas
CREATE TABLE tb_motoristas (
    nome_motorista varchar(60),
    cpf_motorista INT PRIMARY KEY,
    cnh VARCHAR(100),
    cnpj int,
    cidade_motorista VARCHAR(100),
    endereco_motorista VARCHAR(100),
    tel_motorista VARCHAR(15),
    email_motorista VARCHAR(50),
    senha_motorista varchar(50),
    foto_van varchar(50),
    foto_motorista varchar(50),
    valor_cobrado int
);
```

4- Python

O código Python define duas rotas utilizando o framework Flask. A rota principal (/) redireciona o usuário para a página de login, enquanto a rota /logar lida tanto com requisições GET quanto POST. Na requisição GET, o servidor renderiza o template login-motorista.html, que exibe o formulário de login. Quando uma requisição POST é feita, o código extrai as informações de e-mail e senha e as passa para o método logar da classe Usuario. Se o login for bem-sucedido, os dados do motorista são armazenados na sessão e o usuário é redirecionado para a página inicial. Caso contrário, a sessão é limpa e o usuário retorna à página de login.

```
@app.route("/logar", methods=['POST', 'GET'])
def logar():
    if request.method == 'GET':
        return render_template("login-motorista.html")
    else:
        senha = request.form['senha']
        email = request.form['email']
        usuario = Usuario()
        if usuario.logar(email, senha):
            session['usuario_logado'] = {
                "nome": usuario.nome,
                "email": usuario.email,
                "cpf": usuario.cpf
            }
            return redirect('/pag-inicial-motorista')
        else:
            session.clear()
            return redirect("/logar")
```

O método logar da classe Usuario é responsável por autenticar o motorista, conectando-se ao banco de dados por meio de Conexao.conectar(). Ele executa uma consulta SQL para verificar se há um registro correspondente ao e-mail e senha fornecidos. Se for encontrado um resultado, os dados do motorista são armazenados nas propriedades da instância e a variável logado é definida como True. Caso contrário, o login falha e a variável logado

```
def logar(self, email, senha):
        try:
```

```
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            sql = "SELECT nome_motorista, cpf_motorista,
email_motorista FROM tb_motoristas WHERE email_motorista = %s AND
senha_motorista = %s"
            values = (email, senha)
            mycursor.execute(sql, values)

            resultado = mycursor.fetchone()
            if resultado:
                self.nome = resultado[0]
                self.cpf = resultado[1]
                self.email = resultado[2]
                self.logado = True
            else:
                self.logado = False

            mycursor.close()
            mydb.close()
            return self.logado
        except Exception as e:
            print(f"Erro ao logar: {e}")
            return F
```

5-JS

```
<script>
        {% with mensagens = get_flashed_messages() %}
        {% for alerta in mensagens %}
        {{ alerta | safe }}
        {% endfor %}
        {% endwith %}
    </script>
```

APÊNDICE B: Cadastro do motorista

1-HTML

O formulário de cadastro do motorista é uma parte fundamental do sistema de gestão de transporte escolar, projetado para coletar informações essenciais que garantem a segurança e a confiabilidade do serviço. Os dados requisitados incluem nome, CPF, CNH, informações de contato, e endereço. Além disso, o sistema permite o upload de imagens do motorista e da van, assegurando que todos os requisitos de segurança sejam atendidos. As informações coletadas são armazenadas em um banco de dados relacional, que facilita consultas e validações em tempo real.

Quando o motorista preenche o formulário e clica no botão "Cadastrar", um processo de validação de dados é disparado. Isso garante que todas as informações sejam corretas e completas antes de serem registradas no sistema, otimizando a eficiência operacional e aumentando a segurança no

```
<main>
        <div class="titulo">
            <h2><u>Cadastro de Motorista</u></h2>
        </div>

        <form class="container-form" action="#" method="post">
            <section class="itens-form">
                <div class="lado-esquerdo-container">
                    <label for="nome">Nome:</label>
                    <input type="text" id="nome" name="nome"
placeholder="ex: João da Silva" required>

                    <label for="cpf">CPF:</label>
                    <input type="text" id="cpf" name="cpf"
maxlength="14" oninput="formatarCPF(this.value)" placeholder="ex:
000.000.000-00" required>
```

```
                    <label for="cidade">Cidade:</label>
                    <input type="text" id="cidade" name="cidade"
placeholder="ex: Araraquara" required>

                    <label for="telefone">Telefone:</label>
                    <input type="text" id="telefone" name="telefone"
maxlength="15" onkeyup="handlePhone(event)" placeholder="ex: (00) 0000-
0000" required>
                </div>

                <div class="lado-direito-container">
                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email"
placeholder="ex: nome@exemplo.com" required>

                    <label for="endereco">Endereço:</label>
                    <input type="text" id="endereco" name="endereco"
placeholder="ex: Rua das Flores, 123 - Bairro, Cidade, Estado"
required>

                    <label for="senha">Senha:</label>
                    <input type="password" id="senha" name="senha"
minlength="8" placeholder="ex: mínimo 8 caracteres" required>

                    <div class="btn-container">
                        <button type="submit"
class="cadastrar">Cadastrar</button>
                    </div>
                </div>
            </section>
        </form>
    </main>
```

2- CSS

O código CSS aplicado ao formulário estabelece um design limpo e funcional, começando com um reset global que remove margens e preenchimentos indesejados, assegurando uma base uniforme em diferentes navegadores. A tipografia é configurada para garantir legibilidade, e as cores de fundo são escolhidas para criar um ambiente visualmente agradável.

O cabeçalho (header) do sistema incorpora um gradiente linear e sombra, que adicionam profundidade visual, enquanto o elemento principal (main) utiliza flexbox para centralizar o formulário de cadastro na tela. A classe .container-form

é estilizada com max-width e border-radius para garantir uma aparência moderna e acolhedora. As media queries são empregadas para garantir que o layout seja responsivo, ajustando-se de maneira eficiente a diferentes tamanhos de tela e dispositivos.

```
/* Reset básico */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Estilização do corpo */
body {
    font-family: 'Roboto', sans-serif;
    background-color: #ecf0f1; /* Cor de fundo clara */
    color: #2c3e50; /* Cor do texto */
}

/* Estilização do cabeçalho */
header {
    background-color: #2c3e50; /* Azul escuro */
    color: #ecf0f1; /* Texto claro */
    height: 80px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 0 20px;
}

.button-back {
    background-color: transparent;
    border: none;
}

.button-back img {
    width: 20px;
    height: auto;
}

.logo img{
    height: 100px;
}

/* Estilização do main */
main {
```

```
    display: flex;
    justify-content: center;
    flex-direction: column;
    padding: 20px;
}

.titulo{
    text-align: center;
    margin-bottom: 10px;
}

/* Estilização do container-form */
.container-form {
    max-width: 1000px;
    width: 90%;
    margin: 0 auto;
    background-color: #ffffff; /* Cor de fundo do formulário */
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 30px;
}

.container-form h3 {
    text-align: center;
    margin-bottom: 20px;
    color: #E79428; /* Cor do título */
}

/* Layout dos itens do formulário */
.itens-form {
    display: flex;
    gap: 20px;
    flex-wrap: wrap;
}

/* Estilização dos containers de lado */
.lado-esquerdo-container,
.lado-direito-container {
    flex: 1;
    min-width: 250px; /* Largura mínima */
}

/* Estilização dos labels */
.container-form label {
    display: block;
    margin-bottom: 8px;
```

```
    font-weight: 600;
    color: #2c3e50; /* Cor dos labels */
}

/* Estilização dos campos de entrada */
.container-form input {
    width: 100%;
    padding: 12px;
    margin-bottom: 15px;
    border: 1px solid #ddd;
    border-radius: 6px;
    font-size: 16px;
    background-color: #f9f9f9; /* Fundo claro */
}

/* Estilização dos botões */
.btn-container {
    text-align: right;
}

.btn-container button {
    background-color: #E79428; /* Cor do botão */
    color: #ecf0f1; /* Texto claro */
    border: none;
    padding: 12px 24px;
    border-radius: 10px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.btn-container button:hover {
    background-color: #34495e; /* Cor do botão ao passar o mouse */
}

.btn-container button:active {
    transform: scale(0.98);
}

/* Media Queries */

/* Para telas menores que 1200px */
@media (max-width: 1200px) {
    .container-form {
        width: 90%; /* Largura responsiva */
    }
```

```
}

/* Para telas menores que 992px */
@media (max-width: 992px) {
    .itens-form {
        flex-direction: column; /* Coloca os itens em coluna */
    }

    .lado-esquerdo-container,
    .lado-direito-container {
        min-width: 100%; /* Ocupa 100% da largura em dispositivos
pequenos */
    }
}

/* Para telas menores que 768px */
@media (max-width: 768px) {
    .container-form {
        width: 100%; /* Largura total */
    }
}

/* Para telas menores que 576px */
@media (max-width: 576px) {
    .btn-container button {
        width: 100%; /* Botão ocupa toda a largura */
        margin-top: 20px; /* Ajustar margem */
    }
}
```

3- Bancos de dados

O script SQL é responsável pela criação da tabela tb_motoristas, que serve para armazenar todos os dados necessários sobre os motoristas do sistema. A chave primária, definida pela coluna cpf_motorista, assegura que cada motorista tenha um CPF único, evitando duplicidade de registros. A tabela contém campos específicos para identificação, como nome, CPF, CNH e CNPJ, além de dados de contato e localização.

Para garantir a segurança e a integridade dos dados, o campo senha_motorista é incluído para a autenticação do motorista. É importante ressaltar

que, para aumentar a segurança do sistema, recomenda-se o uso de técnicas de hashing para proteger as senhas armazenadas.

```sql
-- Criação da tabela tb_motoristas
CREATE TABLE tb_motoristas (
    nome_motorista varchar(60),
    cpf_motorista VARCHAR(100) PRIMARY KEY,
    cnh VARCHAR(100),
    cnpj VARCHAR(100),
    cidade_motorista VARCHAR(100),
    endereco_motorista VARCHAR(100),
    tel_motorista VARCHAR(15),
    email_motorista VARCHAR(50),
    senha_motorista varchar(50)
);
```

4- Python

A aplicação utiliza o framework Flask para gerenciar as requisições relacionadas ao cadastro de motoristas. A rota definida com o decorador @app.route("/cadastrar-motorista", methods=['GET','POST']) permite tanto a exibição do formulário de cadastro quanto o processamento dos dados enviados pelo usuário.

Quando a requisição é do tipo GET, o servidor renderiza o template cadastro-motorista.html, apresentando o formulário ao usuário. Em caso de uma requisição POST, as informações do formulário são extraídas e um objeto da classe Usuario é criado. O método cadastrar_motorista é chamado para processar o registro. Se o cadastro for bem-sucedido, o usuário é redirecionado para a página de login; caso contrário, ele retorna à página inicial.

```python
@app.route("/cadastrar-motorista", methods=['GET','POST'])
def pag_cadastro_motorista():
    if request.method == 'GET':
        return render_template('cadastro-motorista.html')
    else:
        nome = request.form["nome"]
        cpf = request.form["cpf"]
        cnpj = request.form["cnpj"]
        cnh = request.form["cnh"]
        telefone = request.form["telefone"]
        email = request.form["email"]
        senha = request.form["senha"]
        flash("alert('Usuário cadastrado com sucesso!')")
```

```
        usuario = Usuario()
        if usuario.cadastrar_motorista(nome, cpf, cnpj, cnh, telefone, email,
senha):
            return redirect('/logar')
        else:
            return redirect('/')
```

A classe Usuario encapsula as propriedades do motorista e possui o método cadastrar_motorista, que estabelece uma conexão com o banco de dados. O comando SQL INSERT é utilizado para armazenar os dados do motorista. O uso de um bloco try-except assegura que erros na inserção sejam tratados corretamente, e o método retorna um valor booleano que indica se a operação foi bem-sucedida.

```
class Usuario():
    def __init__(self):
        self.nome = None
        self.cpf = None
        self.email = None
        self.logado = False

    def cadastrar_motorista(self, nome, cpf, cnh, cnpj, telefone, email,
senha):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            sql = f"""INSERT INTO tb_motoristas (nome_motorista,
cpf_motorista, cnh, cnpj, tel_motorista, email_motorista, senha_motorista)
                      VALUES ('{nome}', '{cpf}', '{cnh}', '{cnpj}',
'{telefone}', '{email}', '{senha}');
                    """
            mycursor.execute(sql)
            mydb.commit()
            mycursor.close()
            return True

        except Exception as e:
            print(f"Erro ao cadastrar motorista: {e}")
            return False
```

APÊNDICE C: Pagina Inicial

1-HTML

Este código HTML estrutura uma página inicial para um sistema de transporte escolar com uma barra de navegação, saudação personalizada e uma seção de serviços. A navegação (<nav>) inclui links para acessar o cadastro de alunos, a listagem de passageiros e o histórico de pagamentos, essenciais para o gerenciamento de rotinas dos motoristas. Na seção principal (<main>), uma saudação personalizada exibe o nome do usuário logado através da variável {{ session.usuario_logado.nome }}, seguida de um slogan que destaca os valores de segurança e confiabilidade. Abaixo, uma seção de serviços apresenta três cartões, cada um com um ícone, título e breve descrição. Os cartões destacam as funcionalidades do sistema: o aumento da demanda de clientes, o gerenciamento facilitado dos pagamentos e a promoção de um transporte seguro e confortável para os alunos, facilitando a comunicação e organização dos motoristas com os responsáveis.

```
<nav>
        <a href="/cadastrar-aluno" class="nav-item">Cadastro Alunos</a>
        <a href="/listar-alunos" class="nav-item">Passageiros</a>
        <a href="/historico_pagamento" class="nav-item">Histórico de
Pagamentos</a>
    </nav>

    <main>
        <section class="welcome-section">
            <h1>Olá {{ session.usuario_logado.nome }}!</h1>
            <p class="slogan">Transporte Escolar com Segurança,
Confiabilidade e Praticidade!</p>
        </section>

        <section class="services-section">
            <h2>Serviços Disponíveis</h2>
            <div class="cards-container">
```

```
                <div class="service-card">
                    <img src="/static/icones/icons1-paginicial.png"
alt="Demanda" class="service-icon">
                    <h3>Aumente sua Demanda!</h3>
                    <p>Organize-se e aumente sua quantidade de
clientes.</p>
                </div>

                <div class="service-card">
                    <img src="/static/icones/icons2-paginicial.png"
alt="Gerenciamento" class="service-icon">
                    <h3>Gerencie seus Pagamentos!</h3>
                    <p>Analise seu lucro de forma fácil e
eficiente.</p>
                </div>

                <div class="service-card">
                    <img src="/static/icones/icons3-paginicial.png"
alt="Praticidade" class="service-icon">
                    <h3>Traga Praticidade!</h3>
                    <p>Ofereça passeios com segurança e conforto.</p>
                </div>
            </div>
        </section>
    </main>
```

2-CSS

O CSS define estilos globais para a página, zerando margens e padding e utilizando `box-sizing: border-box` para incluir padding e bordas no cálculo da largura e altura de elementos. O `body` tem um layout flexível (`display: flex`) que ocupa toda a altura da tela, com um fundo cinza claro. O cabeçalho (`header`) utiliza `flexbox` para alinhar itens, apresentando um gradiente de fundo e uma sombra. Os títulos e parágrafos têm estilos específicos, com fontes personalizadas e cores distintas para hierarquia visual. Estilos responsivos são aplicados através de media queries, ajustando o layout de cards e a disposição do cabeçalho para telas menores, garantindo uma experiência de usuário otimizada em dispositivos variados.

```
/* Estilos gerais para zerar margens, padding e ajustar box-sizing */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

```
/* Estilo para o corpo do site */
body {
    font-family: Arial, sans-serif;
    /* Define a fonte do corpo */
    background-color: #e4e4e4;
    /* Cor de fundo neutra */
    color: #fff;
    /* Cor do texto geral branca */
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    min-height: 100vh;
    /* Garante que o body ocupe toda a altura da tela */
}

/* Estilos para o cabeçalho */
header {
    display: flex;
    justify-content: space-between;
    /* Alinha logo e ícone do menu nas extremidades */
    align-items: center;
    /* Centraliza verticalmente os itens no header */
    width: 100%;
    padding: 5px;
    background-image: linear-gradient(to right, #6DA6B1 0%, #346D7E
100%);
    /* Gradiente no background */
    color: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15);
    /* Sombra no header */
    z-index: 999;
    /* Garante que o header fique sempre por cima de outros elementos
*/
}

/* Estilo para a logo */
.logo img {
    width: 150px;
    /* Define o tamanho da logo */
}

/* Ícone de menu responsivo */
.menu-icon {
    cursor: pointer;
    /* Muda o cursor ao passar sobre o ícone */
```

```
    color: #ffcc00;
    /* Define cor do ícone de menu */
    padding-left: 96%;
}

.menu-icon img {
    width: 50px;
    height: 50px;
    /* Aplica um efeito para alterar a cor da imagem usando filtros */
    filter: invert(99%) sepia(1%) saturate(74%) hue-rotate(354deg)
brightness(114%) contrast(100%);
}

/* Estilo do menu lateral oculto inicialmente */

/* Estilos principais do conteúdo */
main {
    padding: 50px;
    max-width: 1200px;
    /* Largura máxima do conteúdo */
    margin: 0 auto;
    /* Centraliza o conteúdo */
    color: #333;
    /* Cor do texto */
}

/* Estilo dos títulos (h2) */
h2 {
    color: #000;
    margin-bottom: 20px;
    font-size: 3rem;
    /* Tamanho maior para destaque */
    font-family: 'Bebas Neue', sans-serif;
    /* Fonte especial */
}

/* Estilo para h4 (subtítulos) */
h4 {
    color: #000;
    margin-bottom: 10px;
    font-family: 'Bebas Neue', sans-serif;
}

/* Margem entre as seções */
section {
    margin-bottom: 40px;
```

```
}

/* Layout de cards para seções de serviços */
.cards2 {
    display: flex;
    justify-content: space-around;
    /* Distribui os cards uniformemente */
    flex-wrap: wrap;
    /* Permite que os cards se ajustem em telas menores */
}

/* Estilo dos cards de serviços */
.card3 {
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 10px;
    width: 250px;
    margin: 20px;
    padding: 20px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    /* Sombra leve */
    text-align: center;
}

/* Estilo dos títulos dos cards */
.card3 h2 {
    color: #6b1c1c;
    /* Cor de destaque */
    font-size: 1.4rem;
    margin-bottom: 15px;
}

/* Estilo dos parágrafos dentro dos cards */
.card3 p {
    color: #333;
    font-size: 0.9rem;
}

/* Estilo dos ícones dentro dos cards */
.icon img {
    width: 60px;
    height: 60px;
    margin-bottom: 15px;
}

/* Estilo do rodapé */
```

```css
footer {
    background-color: #346D7E;
    color: #fff;
    text-align: center;
    padding: 20px 0;
    font-size: 0.9rem;
}

/* Ícones sociais no rodapé */
footer .social-icons {
    margin: 10px 0;
}

/* Estilo dos links de ícones sociais */
footer .social-icons a {
    color: #fff;
    margin: 0 10px;
    text-decoration: none;
    font-size: 1.5rem;
    /* Tamanho dos ícones */
}

/* Estilo do texto do rodapé */
footer p {
    margin-top: 10px;
}

/* Responsividade para telas menores que 768px */
@media (max-width: 768px) {
    .cards2 {
        flex-direction: column;
        /* Alinha os cards em uma coluna */
        align-items: center;
    }

    .card3 {
        width: 80%;
        /* Card ocupa maior parte da largura */
        margin-bottom: 20px;
    }
}

/* Responsividade para telas menores que 600px */
@media (max-width: 600px) {
    main {
        padding: 20px;
```

```
        /* Reduz o padding em telas pequenas */
    }

    header {
        flex-direction: column;
        /* Header em coluna para telas menores */
        padding: 20px;
    }

    h2 {
        font-size: 2rem;
        /* Reduz o tamanho do título */
    }
}
```

3-JS

Este código JavaScript adiciona uma funcionalidade de abrir e fechar menu no site. Quando o conteúdo da página é carregado (`DOMContentLoaded`), o código busca o ícone do menu (`menuIcon`) e o próprio menu (`menu`) pelos seus IDs no HTML. Com isso, ele se prepara para monitorar cliques no ícone.

Sempre que o `menuIcon` é clicado, o código verifica se o menu possui a classe `show`. Caso tenha, o menu é ocultado ao remover a classe `show`; caso contrário, a classe `show` é adicionada, exibindo o menu. Essa alternância permite que o menu deslize para dentro e para fora, conforme o usuário clica no ícone, criando uma experiência interativa e prática.

```
document.addEventListener("DOMContentLoaded", function () {
    const menuIcon = document.getElementById('menu-icon');
    const menu = document.getElementById('menu');

    menuIcon.addEventListener('click', function () {
        // Adiciona ou remove a classe 'show' para mostrar/esconder o
menu
        if (menu.classList.contains('show')) {
            menu.classList.remove('show'); // Esconde o menu (desliza
para fora)
        } else {
            menu.classList.add('show'); // Mostra o menu (desliza para
dentro)
        }
```

```
    });
});


document.addEventListener("DOMContentLoaded", function () {
    const menuIcon = document.getElementById('menu-icon');
    const menu = document.getElementById('menu');

    menuIcon.addEventListener('click', function () {
        // Adiciona ou remove a classe 'show' para mostrar/esconder o
menu
        if (menu.classList.contains('show')) {
            menu.classList.remove('show'); // Esconde o menu (desliza
para fora)
        } else {
            menu.classList.add('show'); // Mostra o menu (desliza para
dentro)
        }
    });
});
```

4- Python

Esta rota define a página inicial do motorista no aplicativo. Quando o usuário acessa `"/pag-inicial-motorista"`, a função `pag_inicial` renderiza o template `"pag-inicial-motorista.html"`, passando a sessão atual (`session`) para o template, permitindo acesso a dados do usuário logado na página.

```
@app.route("/pag-inicial-motorista")
def pag_inicial():
    return render_template("pag-inicial-motorista.html",
session=session)
```

APÊNDICE D: Cadastro do aluno

1-HTML

O formulário de cadastro de alunos é construído utilizando HTML e é projetado para coletar informações essenciais, como nome, telefone e escola. O uso do método POST e o atributo enctype="multipart/form-data" são cruciais para permitir o upload de arquivos, como a foto do aluno. A estrutura do formulário é dividida em duas seções, organizadas por flexbox, garantindo uma disposição responsiva que se adapta a diferentes tamanhos de tela.

Cada campo possui um rótulo (label) associado, o que melhora a acessibilidade para usuários com deficiências. Os inputs incluem atributos como maxlength para limitar a quantidade de caracteres e accept para restringir o tipo de arquivo que pode ser enviado. O botão de submissão, identificado pela classe cadastrar, facilita a estilização e a interação do usuário. Ao enviar o formulário, os dados são estruturados de forma adequada para processamento no servidor.

```
<main>
        <div class="titulo">
            <h2><u>Cadastro de Alunos</u></h2>
        </div>

        <form class="container-form" action="#" method="post"
enctype="multipart/form-data">
            <section class="itens-form">
                <div class="lado-esquerdo-container">
                    <label for="nomeAluno">Nome do aluno:</label>
                    <input type="text" id="nomeAluno" placeholder="ex:
João da Silva" name="nome-aluno" required>

                    <label for="nomeResponsavel">Nome do
responsável:</label>
                    <input type="text" id="nomeResponsavel"
placeholder="ex: Maria de Souza" name="nome-responsavel"
```

```
                        required>

                    <label for="telefoneResponsavel">Telefone do
responsável:</label>
                    <input type="tel" maxlength="15"
onkeyup="handlePhone(event)" id="telefoneResponsavel"
                        placeholder="ex: (00) 0000-0000"
name="telefone-responsavel" required>

                    <label for="escola">Escola:</label>
                    <input type="text" id="escola" placeholder="ex:
Escola Horizonte Azul" name="escola" required>

                </div>

                <div class="lado-direito-container">
                    <label for="emailAluno">Email do
responsável:</label>
                    <input type="email" id="emailAluno"
placeholder="ex: nome@exemplo.com" name="email-aluno" required>

                    <label for="condicaoMedica">Observações:</label>
                    <input type="text" id="condicaoMedica"
placeholder="Insira um texto sobre, se não tem escreva 'Não tem'"
name="condicao-medica" required>

                    <label for="enderecoAluno">Endereço:</label>
                    <input type="text" id="enderecoAluno"
placeholder="ex: Rua das Flores, 123 - Bairro, Cidade, Estado"
                        name="endereco-aluno" required>

                    <label for="fotoAluno">Foto Aluno:</label>
                    <input type="file" id="fotoAluno" accept="image/*"
placeholder="Insira o link da foto aqui"
                        name="foto-aluno" required>

                    <div class="btn-container">
                        <button type="submit"
class="cadastrar">Cadastrar</button>
                    </div>

                </div>
            </section>
        </form>
    </main>
```

2-CSS

O CSS utilizado estabelece um design limpo e moderno para o formulário de cadastro. Um reset global, aplicado pelo seletor universal *, garante que margens e preenchimentos sejam uniformes em todos os navegadores. A tipografia é escolhida para garantir legibilidade, enquanto um esquema de cores e gradientes no header melhora a estética visual.

A estrutura do layout é organizada com flexbox, que permite que os elementos se ajustem de forma responsiva, facilitando a visualização em dispositivos de diferentes tamanhos. Media queries são usadas para adaptar o estilo em diferentes resoluções de tela, alterando propriedades como max-width, padding e tamanho da fonte dos botões. As transições e efeitos aplicados aos botões proporcionam feedback visual ao usuário, melhorando a interação.

```css
/* Reset geral */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Estilização do corpo da página */
body {
    font-family: 'Arial', sans-serif;
    line-height: 1.6;
    background-color: #f4f4f4;
    color: var(--cor-01);
}

header {
    background-image: linear-gradient(to right, var(--cor-05) 0%, var(--cor-
06) 100%);
    /* Gradiente suave de azul claro para azul petróleo */
    color: var(--cor-03);
    /* Texto em bege claro para contraste elegante */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15);
    /* Sombra suave para profundidade */
    height: 90px;
    margin-bottom: 20px;
    display: flex;
    justify-content: space-between;
    align-items: center;
```

```
}

.button-back {
    background-color: transparent;
    border: none;
}

.button-back img {
    width: 20px;
    height: auto;
    margin-left: 30px;
}

figure img {
    width: 100px;
    /* Tamanho da imagem do logo */
    height: auto;
    /* Manter a proporção da imagem */
    margin: 0 20px 0 0;
}

/* Estilização do main */
main {
    display: flex;
    justify-content: center;
    padding: 20px;
}

/* Estilização do container-form */
.container-form {
    max-width: 1000px;
    margin: 0 auto;
    padding: 30px;
    background-color: var(--cor-03);
    border-radius: 10px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}

.container-form h3 {
    text-align: center;
    margin-bottom: 20px;
    color: var(--cor-06);
    font-style: oblique;
}

/* Layout dos itens do formulário */
.itens-form {
    display: flex;
    gap: 20px;
```

```
    flex-wrap: wrap;
    /* Para telas menores, os itens se ajustam */
}

/* Estilização dos containers de lado */
.lado-esquerdo-container,
.lado-direito-container {
    min-width: 100%;
    /* Ocupa 100% da largura em telas pequenas */
}

/* Estilização dos labels */
.container-form label {
    display: block;
    margin-bottom: 8px;
    font-weight: 600;
    color: var(--cor-06);
}

/* Estilização dos campos de entrada */
.container-form input[type="text"],
.container-form input[type="email"],
.container-form input[type="password"],
.container-form input[type="tel"] {
    width: 100%;
    padding: 12px;
    margin-bottom: 15px;
    border: 1px solid #ddd;
    border-radius: 6px;
    font-size: 16px;
    background-color: #f9f9f9;
}

/* Estilização dos botões */
.btn-container {
    text-align: right;
}

.btn-container button {
    background-color: var(--cor-06);
    color: var(--cor-03);
    border: none;
    padding: 12px 24px;
    border-radius: 10px;
    cursor: pointer;
    font-size: 16px;
    margin-top: 40px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}
```

```
.btn-container button:hover {
    background-color: var(--cor-05);
}

.btn-container button:active {
    transform: scale(0.98);
}

/* Media Queries */

/* Small devices (landscape phones, 576px and up) */
@media (min-width: 576px) {
    figure img {
        margin: 0 10px 0 0;
    }

    .container-form {
        max-width: 540px;
        /* Ajuste da largura para dispositivos pequenos */
    }

    .lado-esquerdo-container,
    .lado-direito-container {
        min-width: 100%;
        /* Continua com 100% de largura em dispositivos pequenos */
    }
}

/* Medium devices (tablets, 768px and up) */
@media (min-width: 768px) {
    figure img {
        margin: 0 10px 0 0;
    }

    .container-form {
        max-width: 720px;
    }

    .itens-form {
        flex-wrap: nowrap;
        /* Remove a quebra de linha para tablets */
    }

    .lado-esquerdo-container,
    .lado-direito-container {
        min-width: 300px;
        /* Largura mínima ajustada para tablets */
    }
```

```
}

/* Large devices (desktops, 992px and up) */
@media (min-width: 992px) {
    .container-form {
        max-width: 960px;
    }

    .lado-esquerdo-container,
    .lado-direito-container {
        min-width: 450px;
        /* Ajusta a largura mínima para desktops */
    }
}

/* X-Large devices (large desktops, 1200px and up) */
@media (min-width: 1200px) {
    .container-form {
        max-width: 1000px;
        /* Usa a largura completa para desktops grandes */
    }

    .btn-container button {
        font-size: 18px;
        /* Aumenta o tamanho da fonte do botão */
        padding: 14px 28px;
        /* Aumenta o espaçamento do botão */
    }
}

/* XX-Large devices (larger desktops, 1400px and up) */
@media (min-width: 1400px) {
    .container-form {
        max-width: 1200px;
        /* Aumenta o tamanho máximo para telas muito grandes */
        padding: 50px;
        /* Adiciona mais espaçamento interno */
    }

    .btn-container button {
        font-size: 20px;
        /* Tamanho maior para o botão */
        padding: 16px 32px;
        /* Ajusta o padding do botão */
    }
}
```

3- Banco de dados

O script SQL é responsável pela criação da tabela tb_alunos, que armazena informações dos alunos. A tabela possui um identificador único (id_aluno), que é auto-incrementado, garantindo que cada registro tenha uma chave primária exclusiva. Os campos coletam informações relevantes, como nome, foto, condição médica, e dados do responsável, incluindo telefone, e-mail e endereço.

A relação entre as tabelas é estabelecida pela chave estrangeira (FOREIGN KEY), vinculando o CPF do responsável da tabela tb_responsavel. Essa configuração assegura a integridade referencial, garantindo que cada aluno esteja associado a um responsável válido.

```
-- Criação da tabela tb_alunos
CREATE TABLE tb_alunos (
    id_aluno  int AUTO_INCREMENT,
    nome_aluno VARCHAR(100),
    foto_aluno VARCHAR(300),
    condicao_medica varchar(255),
    escola VARCHAR(100),
    nome_responsavel varchar(100),
    endereco VARCHAR(100),
    telefone_responsavel varchar(100),
    email_responsavel varchar(100),
    idade int,
    cpf_motorista varchar(100),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista),
    PRIMARY KEY (id_aluno)
);
-- Criação da tabela tb_alunos_registrados
CREATE TABLE tb_alunos_clientes (
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);
```

4- Python

O código Python implementa uma rota Flask para o cadastro de alunos, usando o decorador @app.route para lidar com métodos GET e POST. Ao receber uma requisição GET, o servidor renderiza o template cadastro-aluno.html,

apresentando o formulário ao usuário. Na requisição POST, os dados do formulário são extraídos e processados, incluindo o upload da foto do aluno.

Após o upload, o método cadastrar_aluno é chamado para armazenar as informações no banco de dados. Se o cadastro for bem-sucedido, o usuário é redirecionado para a página inicial; em caso de erro, retorna ao formulário de cadastro.

```
@app.route("/cadastrar-aluno", methods=['GET','POST'])
def pag_cadastro_aluno():
    if request.method == 'GET':
        return render_template('cadastro-aluno.html')
    else:
        nome_aluno = request.form["nome-aluno"]
        escola = request.form["escola"]
        foto_aluno = request.files["foto-aluno"]
        condicao_medica = request.form["condicao-medica"]
        nome_responsavel = request.form["nome-responsavel"]
        endereco_responsavel = request.form["endereco-aluno"]
        tel_responsavel = request.form["telefone-responsavel"]
        email_responsavel = request.form["email-aluno"]
        link_foto = upload_file(foto_aluno)
        usuario = Usuario()
        if usuario.cadastrar_aluno(nome_aluno, link_foto, condicao_medica,
escola, nome_responsavel, endereco_responsavel, tel_responsavel,
email_responsavel):
            return redirect('/')
        else:
           return redirect('/cadastrar-aluno')
```

O método cadastrar_aluno é responsável por inserir os dados do aluno no banco de dados. Ele estabelece uma conexão com o banco de dados e executa um comando INSERT para armazenar as informações coletadas. Após a execução do comando, a operação é confirmada com mydb.commit() e o cursor é fechado. O

método retorna um valor booleano indicando o sucesso da operação, permitindo que a função que o chamou tome decisões com base no resultado da inserção.

```
 def cadastrar_aluno(self, nome_aluno, foto_aluno, condicao_medica,
escola, nome_responsavel, endereco_responsavel, tel_responsavel,
email_responsavel):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            sql = f"""
            INSERT INTO tb_alunos (nome_aluno, foto_aluno,
condicao_medica, escola, nome_responsavel, endereco,
telefone_responsavel, email_responsavel)
            VALUES ('{nome_aluno}', '{foto_aluno}',
'{condicao_medica}', '{escola}', '{nome_responsavel}',
'{endereco_responsavel}', '{tel_responsavel}', '{email_responsavel}');
            """

            mycursor.execute(sql)
            mydb.commit()
            mycursor.close()
            return True
        except Exception as e:
            print(f"Erro ao cadastrar aluno: {e}")
            return False
```

APÊNDICE E: Listar aluno

1-HTML

Esse código HTML tem como objetivo criar uma interface de gerenciamento de alunos para um sistema de transporte escolar. Ele começa com uma seção para listar os alunos cadastrados, permitindo a busca e a filtragem por escola. Há uma área para cadastrar novos alunos e uma lista das escolas disponíveis, com a opção de selecionar uma escola específica para exibir os alunos cadastrados nela. Para cada aluno, são exibidas informações como nome, foto, nome e telefone do responsável, além de opções para editar os dados ou excluir o aluno do sistema. A lista de alunos também permite expandir ou recolher detalhes adicionais, como informações de transporte e condição médica.

Além disso, o código inclui um formulário para gerar uma rota de transporte, onde o usuário pode inserir um ponto de partida e gerar um URL da rota. O código utiliza Flask (indicado pelas sintaxes de Jinja2 como `{% for %}` e `{{ }}`) para renderizar dinamicamente a lista de alunos, escolas e outras informações diretamente do servidor. O sistema ainda permite a interação com checkboxes para selecionar alunos, o que pode ser útil para futuras operações de filtragem ou ações em massa. Em suma, o código proporciona uma maneira interativa e organizada para os gestores de transporte escolar administrarem alunos, escolas e rotas, com uma interface focada na usabilidade e no gerenciamento eficiente.

```
<main>
        <section class="c-table">
            <div class="title">
                <h2>Passageiros</h2>
                <a href="/cadastrar-aluno" class="btn-cadaluno"><button> +
Cadastrar Aluno </button></a>
```

```
            </div>

            <div class="container-busca">
                <input type="text" name="query" class="input-busca"
placeholder="Buscar...">
                <button class="icone-busca" type="submit">🔍</button>
            </div>

            <div class="container-listagem-busca">
                <h2>Escolas</h2>
                <ul>
                    <!-- Botão para exibir todos os alunos, aparece apenas
uma vez -->
                    <li>
                        <form method="GET" action="{{
url_for('listar_alunos') }}">
                            <button type="submit" class="btn-escola"
id="btn-todos">Todos</button>
                        </form>
                    </li>

                    <!-- Botões para cada escola, baseados na lista de
escolas -->
                    {% for escola in escolas %}
                    <li>
                        <form method="GET" action="{{
url_for('listar_alunos', escola=escola) }}">
                            <button type="submit" class="btn-escola"
id="btn-{{ escola }}">{{ escola }}</button>

                            <label>
                                <input type="checkbox" class="school-
checkbox" data-address="{{ escola }}"
```

```
                                    value="{{ escola }}">
                                </label>
                            </form>
                        </li>
                        {% endfor %}
                    </ul>
                </div>
            </section>

            {% if alunos %}
            <h2 class="AA">Alunos cadastrados:</h2>

            {% else %}
            <h2 class="AA">Você ainda não possui alunos cadastrados.</h2>
            {% endif %}

            {% for aluno in alunos %}
            <table class="alunos-listados">
                <div class="container-tabela">
                    <tbody>
                        <tr class="aluno">
                            <td>
                                <div class="perfil-aluno">
                                    <div class="foto-aluno">
                                        <img src="{{aluno['foto_aluno']}}"
alt="Foto do Aluno">
                                    </div>
                                    <div class="info-aluno">
                                        <h2> {{aluno['nome_aluno']}} </h2>
                                        <p><strong>Nome do
resposável:</strong> {{aluno['nome_responsavel']}} </p>
                                    <p><strong>Telefone do
resposável:</strong> {{aluno['telefone_responsavel']}} </p>
```

```
                                </div>

                                <div class="container-rota">
                                    <input type="checkbox" class="student-
checkbox" data-address="{{aluno['endereco']}}"
                                           value="{{aluno['nome_aluno']}}">
                                </div>
                            </div>

                            <div class="detalhes" style="display: none;">
                                <div class="info-transporte">

                                    <div class="titulo-editar">
                                        <h3>Informações de Transporte</h3>
                                        <a href="/editar-
aluno/{{aluno['id_aluno']}}"><button class="btn-editar"><img
                                                    class="icon-editar"
src="/static/icones/editar.png"
                                                    alt="Ícone
Editar"></button></a>
                                    </div>

                                    <p><strong>Endereço:</strong>
{{aluno['endereco']}} </p>
                                    <p><strong>Escola:</strong>
{{aluno['escola']}} </p>
                                </div>

                                <div class="info-extra">
                                    <h3>Condição Médica</h3>
                                    <p> {{aluno['condicao_medica']}} </p>
                                </div>
```

```
                                <div class="container-btn-excluir">
                                    <a href="/quebra-
contrato/{{aluno['id_aluno']}}"><button>Excluir Aluno</button></a>
                                </div>
                            </div>

                            <div class="button-container">
                                <button class="btn-expandir"
onclick="toggleExpandir(this)">▼</button>
                            </div>
                        </td>
                    </tr>
                </tbody>
            </div>
        </table>
        {% endfor %}
        <section class="rota">
            <label for="origin">Ponto de Partida:</label>
            <br>
            <input type="text" id="origin" placeholder="Insira o ponto de
partida">

            <button id="generateRoute">Gerar URL da Rota</button>

            <div id="routeUrl"></div>
        </section>
    </main>
```

2-CSS

O estilo da página é elaborado com CSS, priorizando uma interface visual atraente e fácil de usar. O layout é projetado para ser fluido e responsivo, ajustando-se de forma eficiente a diversas resoluções de tela, garantindo que a usabilidade não seja comprometida em dispositivos diferentes. Elementos essenciais, como

tabelas e botões, são cuidadosamente estilizados com bordas arredondadas e sombras sutis, o que não apenas melhora a estética, mas também ajuda a destacar as interações do usuário.

Além disso, os detalhes de cada aluno são inicialmente ocultos, mas podem ser revelados com transições suaves, proporcionando uma experiência dinâmica e envolvente. Essa abordagem visualmente agradável contribui para a fluidez da navegação na página, enquanto o CSS também cuida da apresentação do campo de busca e dos botões, assegurando que a interação do usuário seja intuitiva e atraente. O design é pensado para equilibrar funcionalidade e estética, facilitando o acesso às informações de forma agradável e eficaz.

```css
* {
    box-sizing: border-box;
    padding: 0;
    margin: 0;
}

body {
    font-family: 'Arial', sans-serif;
    line-height: 1.6; /* Melhora a legibilidade */
    background-color: var(--cor-04); /* Cor de fundo clara */
    color: var(--cor-01); /* Cor do texto */
}

header {
    background-color: #2c3e50;
    height: 80px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    width: 100%;
```

```
}

.button-back {
    background-color: transparent;
    border: none;
    cursor: pointer;
    transition: transform 0.3s ease;
}

.button-back img {
    width: 20px;
    height: auto;
}

.button-back:hover {
    transform: scale(1.1);
}

.logo img {
    height: 30px;
}

main {
    width: 100%;
    margin: auto;
}

.c-table {
    display: flex;
    margin: 20px 0;
    justify-content: space-between;
    align-items: center;
    padding: 0 30px;
```

```
}

.btn-cadaluno button {
    background-color: #e79428;
    color: var(--cor-03);
    font-size: 15px;
    border: none;
    padding: 7px 12px;
    border-radius: 50px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    margin-bottom: 80px;
}

.btn-cadaluno button:hover {
    background-color: #b06d14;
}

/* Estilo do container de busca */
.container-busca {
    display: flex;
    justify-content: center;
    margin-top: 20px;
    margin-bottom: 20px;
    margin-left: 45px;
}

.input-busca {
    border: none;
    padding: 10px 15px;
    outline: none;
    border-radius: 25px 0 0 25px;
}
```

```
.icone-busca {
    background-color: #2c3e50;
    border: none;
    padding: 8px 13px;
    cursor: pointer;
    border-radius: 0 25px 25px 0;
}

.icone-busca:hover {
    background-color: #456789;
}

.container-tabela {
    margin-top: 2rem;
}

.alunos-listados {
    background-color: var(--cor-03);
    margin: 0 auto;
    width: 80%;
    max-width: 900px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 20px;
    margin-bottom: 20px;
}

td {
    padding: 12px 20px;
    font-weight: bold;
    position: relative; /* Necessário para posicionar o botão corretamente
*/
```

```
}

.perfil-aluno {
    display: flex;
    align-items: center;
    margin-bottom: 20px;
}

.foto-aluno {
    width: 120px; /* ou ajuste conforme necessário */
    height: 120px; /* ou ajuste conforme necessário */
    overflow: hidden; /* Garante que a imagem não ultrapasse o contêiner
*/
    border-radius: 50%; /* Garante que o contêiner também seja redondo */
    border: 2px solid #e79428;
}

.foto-aluno img {
    width: 100%; /* Faz com que a imagem preencha o contêiner */
    height: auto; /* Mantém a proporção */
    display: block; /* Remove espaços indesejados */
}

.info-aluno {
    margin-left: 20px;
}

.info-aluno h2 {
    font-size: 24px;
    margin: 0;
    color: var(--cor-01);
}
```

```
.info-aluno p {
    font-size: 16px;
    color: var(--cor-02);
}

/* Detalhes do Aluno */
.detalhes {
    display: none; /* Esconde inicialmente */
    background-color: #F2F2F2; /* Cor de fundo cinza claro */
    padding: 10px;
    border-radius: 4px;
    margin-top: 10px; /* Espaço entre o cabeçalho e a área expandida */
    transition: max-height 0.3s ease, padding 0.3s ease;
}

.btn-expandir {
    position: absolute; /* Posiciona o botão em relação ao contêiner do
aluno */
    right: 0; /* Ajuste a distância da borda direita */
    top: 0;
    background-color: transparent; /* Remova a cor de fundo se necessário
*/
    border: none; /* Remove a borda padrão */
    cursor: pointer; /* Muda o cursor para mão */
    font-size: 1rem; /* Tamanho do ícone */
    color: #e79428; /* Cor do texto */
}

.info-transporte {
    margin-bottom: 20px;
}

.info-transporte h3 {
```

```
    font-size: 20px;
    margin-bottom: 10px;
    color: #e79428;
}

.info-transporte p {
    font-size: 16px;
    color: var(--cor-01);
    margin-bottom: 5px;
}

.titulo-editar {
    display: flex;
    justify-content: space-between;
}

.btn-editar {
    background-color: #2c3e50;
    border: none;
    border-radius: 50%; /* Faz o botão ser redondo */
    width: 30px; /* Ajuste conforme necessário */
    height: 30px; /* Ajuste conforme necessário */
    display: flex;
    justify-content: center;
    align-items: center;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.btn-editar:hover {
    background-color: #456789; /* Cor ao passar o mouse */
}
```

```
.icon-editar {
    width: 20px; /* Tamanho do ícone */
    height: 20px; /* Tamanho do ícone */
}

.info-extra h3 {
    font-size: 20px;
    margin-bottom: 10px;
    color: #e79428;
}

.container-btn-excluir {
    margin-left: 87%;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.container-btn-excluir button {
    background-color: #e79428; /* Cor de fundo */
    color: var(--cor-03); /* Cor do texto */
    border: none; /* Remove borda padrão */
    padding: 12px; /* Espaçamento interno */
    border-radius: 50px; /* Bordas arredondadas */
    cursor: pointer; /* Cursor de mão */
    transition: background-color 0.3s ease, transform 0.2s ease; /*
Transições suaves */
}

.container-btn-excluir button:hover {
    background-color: #b06d14;
    transform: translateY(-2px);
}

/* Estilo do título de alunos */
```

```
.AA {
    color: #333;
    font-size: 24px;
    text-align: center;
    margin-bottom: 20px;
}

/* Estilo do campo "Ponto de Partida" e botão de rota */
.route-container {
    display: flex;
    align-items: center;
    gap: 10px;
    margin-top: 20px;
}

#origin {
    padding: 10px;
    font-size: 14px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 300px;
}

#generateRoute {
    padding: 10px 15px;
    font-size: 14px;
    background-color: #2c3e50;
    color: #fff;
    border: none;
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s;
}
```

```
#generateRoute:hover {
    background-color: #456789;
}

.rota {
    padding: 20px;
}

form {
    display: flex;
    align-items: center;
    padding-right: 3px;
}

form label {
    margin-top: 7px;
}

.container-listagem-busca h2 {
    font-size: 18px;
    color: #2c3e50;
    text-align: center;
}

.container-listagem-busca ul {
    list-style-type: none;
}

.container-listagem-busca li {
    margin-bottom: 5px;
}
```

```
.container-listagem-busca .btn-escola {
    background-color: #e79428;
    color: #fff;
    padding: 8px 12px;
    border: none;
    border-radius: 20px;
    width: 100%;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    text-align: center;
    margin-right: 5px;
}

.container-listagem-busca .btn-escola:hover {
    background-color: #b06d14;
}

/* Estilo do container de rota */
.container-rota {
    display: flex;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    margin: 20px auto;
}

#origin {
    border-radius: 5px;
    border: 1px solid #ccc;
}

#generateRoute {
    background-color: #2c3e50;
```

```
    color: #fff;
    padding: 10px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

#generateRoute:hover {
    background-color: #456789;
}

/* Estilo do checkbox */
input[type="checkbox"] {
    appearance: none; /* Remove a aparência padrão do checkbox */
    width: 20px;
    height: 20px;
    border: 2px solid #ccc; /* Cor da borda desmarcada */
    border-radius: 50%; /* Torna o checkbox redondo */
    position: relative;
    cursor: pointer;
    outline: none;
    transition: background-color 0.3s ease, border-color 0.3s ease;
}

input[type="checkbox"]:checked {
    background-color: #e79428; /* Cor de fundo ao marcar */
    border-color: #e79428; /* Borda fica da mesma cor */
}

/* Estilo opcional para o ícone de verificação */
input[type="checkbox"]:checked::after {
    content: '✔'; /* Ícone de verificação */
    font-size: 14px;
```

```
    color: white;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    line-height: 1;
}
```

3-Banco de dados

A tabela `tb_alunos` desempenha um papel crucial no armazenamento de informações detalhadas sobre os alunos, incluindo dados como nome, foto, condição médica e idade. Cada aluno é identificado de forma única por meio de um campo `id_aluno`, que é gerado automaticamente pelo sistema, garantindo a singularidade dos registros. O campo `responsavel_aluno` estabelece uma conexão direta com a tabela `tb_responsavel`, utilizando uma chave estrangeira que associa cada aluno ao seu respectivo responsável, identificado por meio do CPF.

Essa estrutura não apenas assegura a integridade dos dados, mas também facilita um gerenciamento eficiente de informações essenciais para o transporte escolar. Para otimizar ainda mais essa relação, a tabela `tb_alunos_clientes` é utilizada para vincular os alunos aos motoristas responsáveis pelo transporte, criando uma rede clara e organizada que permite um acompanhamento efetivo das atividades escolares. Essa interconexão entre tabelas é fundamental para garantir que todas as partes envolvidas no transporte escolar estejam devidamente alinhadas e acessíveis, promovendo uma experiência mais eficiente e segura para alunos e responsáveis.

```
CREATE TABLE tb_alunos (
    id_aluno  int AUTO_INCREMENT,
    nome_aluno VARCHAR(100),
    foto_aluno VARCHAR(300),
    condicao_medica varchar(300),
    idade int,
    responsavel_aluno int,
    FOREIGN KEY (responsavel_aluno) REFERENCES
tb_responsavel(cpf_responsavel),
```

```
    PRIMARY KEY (id_aluno)
);
-- Criação da tabela tb_alunos_registrados
CREATE TABLE tb_alunos_clientes (
    id_aluno INT,
    cpf_motorista int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES
tb_motoristas(cpf_motorista)
);
```

4-Python

A rota `/listar-alunos` desempenha um papel crucial no gerenciamento das requisições GET e POST, sendo fundamental para a apresentação da lista de alunos cadastrados no sistema. Quando uma requisição GET é realizada, o sistema inicia um processo de verificação para determinar se o usuário está autenticado, utilizando a chave `usuario_logado` armazenada na sessão do navegador. Se a autenticação for bem-sucedida, o código busca um parâmetro de consulta denominado `query` na URL. Esse parâmetro pode ser utilizado para filtrar a lista de alunos, chamando o método `pesquisar_aluno` da classe `Usuario`, que retorna apenas os alunos que atendem aos critérios especificados.

Caso nenhum parâmetro de busca seja fornecido, o sistema opta por invocar o método `listar_aluno`, que recupera todos os alunos cadastrados sem aplicar quaisquer filtros. Após a coleta dos dados, a lista de alunos é então passada para o template `listar-aluno.html`. Esse template é responsável por renderizar as informações em uma interface visual organizada, exibindo detalhes importantes, como nome, foto, condição médica e outros dados relevantes. Dessa forma, a rota não apenas facilita a busca filtrada de alunos, mas também permite uma exibição abrangente e interativa das informações, tornando a gestão da lista de alunos mais acessível e eficiente para os usuários.

```
app.route("/listar-alunos", methods=['GET', 'POST'])
def listar_alunos():
    if request.method == 'GET':
```

```
        if 'usuario_logado' in  session:
            usuario = Usuario()
            query = request.args.get('query')
            if query:
                lista_alunos = usuario.pesquisar_aluno(query)
            else:
                lista_alunos = usuario.listar_aluno()

            return render_template("listar-aluno.html",
alunos=lista_alunos)
        else:
            return redirect('/logar')
```

O método `listar_aluno` inicia estabelecendo uma conexão com o banco de dados via `Conexao.conectar()`. Ele executa uma consulta SQL SELECT que recupera dados de todos os alunos da tabela `tb_alunos`, como nome, foto e condição médica.

Os resultados da consulta são processados em um loop, onde cada registro é transformado em um dicionário e armazenado em uma lista. Após a coleta das informações, a conexão com o banco de dados é encerrada e a lista de alunos é retornada. Se ocorrer algum erro durante o processo, o método retorna `False`.

```
 def listar_aluno(self):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            sql = f"SELECT nome_aluno, foto_aluno, condicao_medica,
escola, telefone_responsavel, nome_responsavel, endereco, id_aluno
FROM tb_alunos"

            mycursor.execute(sql)
            resultados = mycursor.fetchall()
            alunos = []
            for linha in resultados:
```

```
            alunos.append({"nome_aluno":linha[0],
                           "foto_aluno": linha[1],
                           "condicao_medica": linha[2],
                           "escola": linha[3],
                           "telefone_responsavel": linha[4],
                           "nome_responsavel": linha[5],
                           "endereco": linha[6],
                           "id_aluno": linha[7]
            })

        mydb.close()
        return alunos
    except:
        return False
```

5-JavaScript

Este código JavaScript adiciona interatividade aos botões de filtro de escolas na página. Quando o documento é carregado, o script seleciona todos os botões com IDs que começam com "btn-" e associa um evento de clique a cada um deles. Quando um botão é clicado, o script impede o comportamento padrão do navegador (evitando o recarregamento da página) e verifica se o botão clicado é o "Todos" (ID `btn-todos`). Caso seja, o usuário é redirecionado para a página `/listar-alunos`, que exibe todos os alunos sem qualquer filtro. Se o botão for de uma escola específica, o nome da escola é extraído do ID do botão e o usuário é redirecionado para uma URL filtrada com o parâmetro da escola na consulta GET.

O uso de `encodeURIComponent` assegura que o nome da escola seja corretamente formatado na URL, mesmo que contenha caracteres especiais. Esse processo permite uma navegação mais fluida entre as listas de alunos, proporcionando uma experiência dinâmica e sem a necessidade de recarregar a página. Em suma, o script facilita a aplicação de filtros de escolas de maneira simples e eficiente, mantendo a usabilidade da interface e melhorando a experiência do usuário.

```
document.addEventListener("DOMContentLoaded", function() {
    // Obtém todos os botões de escolas usando a classe 'escola-btn'
    const escolaButtons = document.querySelectorAll("button[id^='btn-']");

    // Adiciona um event listener para cada botão individualmente
    escolaButtons.forEach(function(button) {
        button.addEventListener("click", function(event) {
            // Evita comportamento padrão para que possamos controlar o
evento
            event.preventDefault();

            // Verifica se o botão clicado é o "Todos" (sem filtro)
            if (this.id === "btn-todos") {
                // Redireciona para listar todos os alunos
                window.location.href = `/listar-alunos`;
            } else {
                // Para botões de escola, extrai o nome da escola a partir
do ID do botão
                const escola = this.id.replace("btn-", "");

                // Redireciona para a URL com o filtro da escola, usando o
parâmetro GET
                window.location.href = `/listar-
alunos?escola=${encodeURIComponent(escola)}`;
            }
        });
    });
});
```

O script define a função `toggleExpandir`, que alterna a visibilidade dos detalhes de um aluno ao clicar em um botão. Quando o botão é pressionado, ele encontra o container de detalhes (`.detalhes`) associado à linha do aluno e alterna seu estilo de exibição entre `block` (visível) e `none` (oculto). A função também

altera o texto do botão, exibindo uma seta para cima (`▲`) quando os detalhes estão visíveis, e uma seta para baixo (`▼`) quando estão ocultos. Isso proporciona uma experiência interativa, permitindo expandir ou recolher informações adicionais sobre cada aluno.

```
<script>
        function toggleExpandir(button) {
            const alunoRow = button.closest('.aluno');
            const detalhesContainer = alunoRow.querySelector('.detalhes');

            const isHidden = detalhesContainer.style.display === 'none';
            detalhesContainer.style.display = isHidden ? 'block' : 'none';
            button.textContent = isHidden ? '▲' : '▼';
        }

    </script>
```

APÊNDICE F: Detalhar aluno

1-HTML

O trecho de código HTML define uma seção oculta chamada `detalhes`, destinada a exibir informações adicionais sobre um aluno de forma organizada. Inicialmente invisível, essa seção pode ser revelada através de interações no front-end, oferecendo uma experiência de usuário dinâmica. Dentro dela, a subseção `info-transporte` apresenta campos com o endereço do aluno e a escola que ele frequenta, facilitando o acesso a informações essenciais de transporte.

A subseção `info-extra` destaca a condição médica do aluno, permitindo que os responsáveis identifiquem rapidamente dados críticos. Além disso, a seção `container-btn-excluir` inclui um botão para excluir o aluno, possibilitando ações diretas na interface. Essa estrutura melhora a usabilidade da página, garantindo que detalhes relevantes sejam exibidos apenas quando necessário.

```
<!-- Container para informações adicionais, inicialmente ocultas -->
                                <div class="detalhes"
style="display: none;">
```

```
                                    <!-- Informações de transporte -
->
                                    <div class="info-transporte">
                                        <h3>Informações de
Transporte</h3>

<p><strong>Endereço:</strong> {{}} </p> <!-- Endereço do aluno -->
                                        <p><strong>Escola:</strong>
{{}} </p> <!-- Escola que o aluno frequenta -->
                                    </div>

                                    <!-- Informações extras do aluno
-->
                                    <div class="info-extra">
                                        <h3>Condição Médica</h3>
                                        <p> {{}} </p> <!-- Qualquer
informação adicional relevante -->
                                    </div>

                                    <div class="container-btn-
excluir">
                                        <button>Excluir
Aluno</button>
                                    </div>
                                </div>
-- Criação da tabela tb_alunos_registrados
```

2-CSS

Esse código CSS define o estilo visual da interface de uma aplicação que lista informações sobre alunos, utilizando a fonte 'Arial' e uma paleta de cores gerida por variáveis para garantir uma aparência moderna e legível. O layout é organizado com flexbox, o que proporciona uma apresentação responsiva e adaptável dos elementos, como o campo de busca e a tabela de alunos.

A seção de detalhes dos alunos, que pode ser expandida ou oculta, é estilizada para ser visualmente atraente e fácil de interagir, com transições suaves que melhoram a experiência do usuário. Botões de ação, como o de exclusão, são destacados, promovendo uma melhor usabilidade da interface ao facilitar a interação do usuário com as funcionalidades disponíveis.

```
body {
```

```
    font-family: 'Arial', sans-serif; /* Fonte limpa e moderna */
    line-height: 1.6; /* Melhora a legibilidade */
    background-color: var(--cor-04); /* Cor de fundo clara */
    color: var(--cor-01); /* Cor do texto */
}

main {
    width: 1200px;
    margin: auto;
}

.c-table {
    display: flex;
    margin-bottom: 20px;
    justify-content: space-between;
    align-items: center;
}

.container-busca {
    display: flex;
    align-items: center;
    position: relative; /* Para posicionar o botão corretamente */
}

.input-busca {
    padding: 10px 40px 10px 12px; /* Adiciona espaço para o ícone */
    border: 1px solid #ccc;
    border-radius: 10px;
    font-size: 14px;
    width: 100%; /* Preencher o espaço disponível */
    transition: border-color 0.3s; /* Efeito de transição */
}

.input-busca:focus {
    border-color: var(--cor-06); /* Cor ao focar */
    outline: none; /* Remove contorno padrão */
}

.icone-busca {
    position: absolute;
    right: 10px; /* Espaço à direita */
    background: transparent; /* Fundo transparente */
    border: none; /* Remove borda */
    cursor: pointer; /* Mão ao passar o mouse */
    font-size: 16px; /* Tamanho do ícone */
}
```

```
.icone-busca:hover {
    color: var(--cor-06); /* Muda a cor ao passar o mouse */
}

.alunos-listados {
    background-color: var(--cor-03);
    margin: 0 auto;
    width: 80%;
    max-width: 900px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 20px;
}

.alunos-listados td {
    position: relative;
}

.alunos-listados td .button-container {
    position: absolute;
    right: 0;
    top: 15%;
    transform: translateY(-50%);
}

td {
    padding: 12px 20px;
    font-weight: bold;
}

.perfil-aluno {
    display: flex;
    align-items: center;
    margin-bottom: 20px;
}

.foto-aluno img {
    width: 120px;
    height: 120px;
    border-radius: 50%;
    object-fit: cover;
    border: 3px solid var(--cor-06);
}

.info-aluno {
```

```
    margin-left: 20px;
}

.info-aluno h2 {
    font-size: 24px;
    color: var(--cor-06);
    margin: 0;
    padding: 0;
}

.info-aluno p {
    font-size: 16px;
    color: var(--cor-02);
}

.info-aluno {
    margin-left: 20px;
}

.info-aluno h2 {
    font-size: 24px;
    color: var(--cor-01);
}

.info-aluno p {
    font-size: 16px;
    color: var(--cor-02);
}

/* Remover a repetição e manter o estilo dos detalhes */
.detalhes {
    display: none; /* Esconde inicialmente */
    background-color: #F2F2F2; /* Cor de fundo cinza claro */
    padding: 10px;
    border-radius: 4px;
    margin-top: 10px; /* Espaço entre o cabeçalho e a área expandida
*/
    transition: max-height 0.3s ease, padding 0.3s ease;
}

.info-transporte {
    margin-bottom: 20px;
}

.info-transporte h3 {
    font-size: 20px;
```

```
    margin-bottom: 10px;
    color: var(--cor-06);
}

.info-transporte p {
    font-size: 16px;
    color: var(--cor-01);
    margin-bottom: 5px;
}

.info-extra h3 {
    font-size: 20px;
    margin-bottom: 10px;
    color: var(--cor-06);
}

/* Ajuste do botão de expansão para manter a seta no mesmo lugar */
.btn-expandir {
    position: absolute;
    right: 0;
    background-color: transparent;
    color: var(--cor-06);
    border: none;
    cursor: pointer;
    font-size: 0.9rem;
}

.container-btn-excluir {
    margin-left: 87%;
}

.container-btn-excluir button {
    background-color: var(--cor-06); /* Cor de fundo */
    color: var(--cor-03); /* Cor do texto */
    border: none; /* Remove borda padrão */
    padding: 12px; /* Espaçamento interno */
    border-radius: 6px; /* Bordas arredondadas */
    cursor: pointer; /* Cursor de mão */
    transition: background-color 0.3s ease, transform 0.2s ease; /*
Transições suaves */
}
```

3-Bancos de dados

A tabela `tb_alunos` é responsável por armazenar informações detalhadas sobre os alunos, incluindo um identificador único (`id_aluno`), nome, foto, condição

médica e idade. O campo `responsavel_aluno` estabelece uma relação direta entre cada aluno e seu responsável, utilizando uma chave estrangeira que referencia a tabela `tb_responsavel`, garantindo que todos os responsáveis estejam devidamente registrados no sistema.

A implementação do recurso `AUTO_INCREMENT` para o campo `id_aluno` assegura que os IDs sejam gerados de forma automática e única, evitando duplicidades. Essa estrutura organizacional permite um gerenciamento eficaz e integrado dos dados dos alunos, facilitando o acesso e a manutenção das informações necessárias para o transporte escolar. Dessa forma, a tabela contribui para a eficiência do sistema, promovendo a integridade e a confiabilidade dos dados.

```
CREATE TABLE tb_alunos (
    id_aluno  int AUTO_INCREMENT,
    nome_aluno VARCHAR(100),
    foto_aluno VARCHAR(300),
    condicao_medica varchar(300),
    idade int,
    responsavel_aluno int,
    FOREIGN KEY (responsavel_aluno) REFERENCES
tb_responsavel(cpf_responsavel),
    PRIMARY KEY (id_aluno)
);
CREATE TABLE tb_alunos_clientes (
    id_aluno INT,
    cpf_motorista int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES
tb_motoristas(cpf_motorista)
);
```

4-Python

A rota /listar-alunos é responsável por exibir a lista de alunos em uma aplicação web. Quando acessada, a função listar_alunos é chamada. Esta função aceita requisições GET e POST, mas processa apenas as GET.

Dentro da função, um objeto da classe Usuario é criado para acessar o método listar_aluno, que recupera os dados dos alunos do banco de dados, como nome e informações dos responsáveis. Os dados obtidos são passados para o template listar-aluno.html, onde são renderizados para visualização. Essa estrutura garante que a apresentação dos dados esteja separada da lógica de negócios, promovendo uma arquitetura de software mais limpa e eficiente.

```
@app.route("/listar-alunos", methods=['GET', 'POST'])
def listar_alunos():
    if request.method == 'GET':
        usuario = Usuario()
        lista_alunos = usuario.listar_aluno()
        return render_template("listar-aluno.html", alunos=lista_alunos)
```

O método listar_aluno na classe Usuario tem a função de recuperar as informações dos alunos no banco de dados. Ele começa estabelecendo uma conexão através do método Conexao.conectar(), que é essencial para interagir com os dados.

Uma consulta SQL é então executada para selecionar informações relevantes da tabela tb_alunos. Os resultados são formatados em dicionários e armazenados em uma lista chamada alunos. Após o processamento, a conexão com o banco de dados é fechada, e a lista é retornada. Se ocorrer um erro, o método retorna False, indicando falha na operação. Essa estrutura assegura uma recuperação de dados eficiente e um manejo de erros robusto.

```
def listar_aluno(self):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            sql = f"SELECT nome_aluno, foto_aluno, condicao_medica, escola,
telefone_responsavel, nome_responsavel, endereco, id_aluno FROM tb_alunos"

            mycursor.execute(sql)
            resultados = mycursor.fetchall()
            alunos = []
            for linha in resultados:
                alunos.append({"nome_aluno":linha[0],
                               "foto_aluno": linha[1],
```

```
                                "condicao_medica": linha[2],
                                "escola": linha[3],
                                "telefone_responsavel": linha[4],
                                "nome_responsavel": linha[5],
                                "endereco": linha[6],
                                "id_aluno": linha[7]

                            })

                mydb.close()
                return alunos
            except:
                return False
```

5- JS

O script define funções para melhorar a interação do usuário com a interface da aplicação, focando na exibição de detalhes dos alunos e na navegação do menu. A função toggleExpandir(button) alterna a visualização dos detalhes do aluno correspondente ao botão clicado. Usando o método closest(), a função identifica a linha do aluno e muda a propriedade display do container de detalhes entre 'none' e 'block', enquanto o texto do botão muda entre uma seta para cima e uma seta para baixo, proporcionando feedback visual ao usuário.

O evento DOMContentLoaded garante que o código seja executado somente após o carregamento completo do DOM, prevenindo erros. Um EventListener no ícone do menu alterna a classe 'show', permitindo mostrar ou ocultar um menu responsivo. O botão de alternância (toggleBtn) manipula o display do menu menu1 entre 'grid' e 'none', facilitando a navegação.

```
<script>
    // Função para expandir/ocultar detalhes do aluno
    function toggleExpandir(button) {
        // Encontra a linha do aluno correspondente ao botão clicado
        const alunoRow = button.closest('.aluno');
        // Seleciona o container de detalhes do aluno
        const detalhesContainer = alunoRow.querySelector('.detalhes');

        // Alterna entre mostrar e ocultar os detalhes
        if (detalhesContainer.style.display === 'none') {
            detalhesContainer.style.display = 'block';  // Mostra os detalhes
            button.textContent = '▲';  // Muda o ícone do botão para uma seta
para cima
```

```
        } else {
            detalhesContainer.style.display = 'none';  // Oculta os detalhes
            button.textContent = '▼';  // Muda o ícone do botão para uma seta
para baixo
        }
    }

    document.addEventListener("DOMContentLoaded", function() {
        const menuIcon = document.getElementById('menu-icon');
        const menu = document.getElementById('menu');

        menuIcon.addEventListener('click', function() {
            // Adiciona ou remove a classe 'show' para mostrar/esconder o menu
            if (menu.classList.contains('show')) {
                menu.classList.remove('show'); // Esconde o menu (desliza para
fora)
            } else {
                menu.classList.add('show'); // Mostra o menu (desliza para
dentro)
            }
        });
    });

    // Função para abrir e fechar o menu
    document.getElementById('toggleBtn').addEventListener('click', function()
{
        var menu = document.getElementById('menu1');
        menu.style.display = menu.style.display === 'grid' ? 'none' : 'grid';
        this.innerHTML = menu.style.display === 'grid' ? '&#9660;' :
'&#9650;';
    });


    const details = document.getElementById('details');
</script>
```

APÊNDICE G: Rotas

Este código JavaScript adiciona funcionalidade a um botão que, ao ser clicado, gera uma URL do Google Maps com uma rota personalizada. Ele começa coletando os endereços das escolas e alunos selecionados por meio de caixas de seleção (checkboxes). Para isso, ele busca todos os checkboxes marcados com a classe `.school-checkbox` para escolas e com a classe `.student-checkbox` para alunos, armazenando os endereços em dois arrays: `selectedSchools` e `selectedStudents`. Também é coletado o ponto de partida inserido pelo usuário no campo de texto com o ID `origin`. Se houver pelo menos uma escola ou aluno selecionado e o ponto de partida não estiver vazio, o código continua para gerar a URL da rota.

A URL gerada é uma consulta para o Google Maps, configurada com o ponto de partida, o destino (definido como o último endereço da lista de escolas e alunos) e waypoints (os endereços intermediários, ou seja, todos os endereços selecionados, exceto o destino final). A URL é então exibida na página como um link clicável, que ao ser pressionado, abre o Google Maps em uma nova aba com a rota traçada. Caso as condições não sejam atendidas (nenhuma escola ou aluno selecionado, ou o campo de origem vazio), um alerta é exibido solicitando que o usuário preencha as informações corretamente.

```
document.getElementById('generateRoute').addEventListener('click',
function() {
    const selectedSchools = [];
    const selectedStudents = [];

    // Coletar endereços das escolas selecionadas
    const schoolCheckboxes = document.querySelectorAll('.school-
checkbox:checked');
```

```
    schoolCheckboxes.forEach(checkbox => {
        selectedSchools.push(checkbox.getAttribute('data-address'));
    });

    // Coletar endereços dos alunos selecionados
    const studentCheckboxes = document.querySelectorAll('.student-
checkbox:checked');
    studentCheckboxes.forEach(checkbox => {
        selectedStudents.push(checkbox.getAttribute('data-address'));
    });

    const originInput = document.getElementById('origin');
    const origin = originInput.value.trim();

    // Verifica se há escolas ou alunos selecionados
    if ((selectedSchools.length > 0 || selectedStudents.length > 0) &&
origin !== '') {
        // Combina as escolas e alunos em um único array
        const allSelectedAddresses = [...selectedSchools,
...selectedStudents];

        // Define o destino como o último endereço selecionado
        const destination =
allSelectedAddresses[allSelectedAddresses.length - 1];

        // Define os waypoints como todos os endereços, exceto o último
        const waypoints = allSelectedAddresses.slice(0, -1).join('|');

        // Gera a URL do Google Maps
        const routeUrl =
`https://www.google.com/maps/dir/?api=1&origin=${encodeURIComponent(origin
)}&destination=${encodeURIComponent(destination)}&waypoints=${encodeURICom
ponent(waypoints)}`;
```

```
        // Atualiza o HTML com o link da rota
        document.getElementById('routeUrl').innerHTML = `<a
href="${routeUrl}" target="_blank">Abrir Rota no Google Maps</a>`;
    } else {
        alert('Por favor, selecione pelo menos uma escola ou um aluno, e
insira o ponto de partida.');
    }
});
```

APÊNDICE G: Editar perfil

1-HTML

Este código é um formulário HTML para editar as informações de um aluno no sistema. O formulário é dividido em duas seções: a parte esquerda contém campos para editar o nome, o nome do responsável, telefone, escola e e-mail, enquanto a parte direita permite editar as observações sobre o aluno, endereço e foto. O formulário é enviado via método POST para a URL `/editar-aluno/{{ aluno['id_aluno'] }}`, onde o ID do aluno é dinamicamente inserido a partir de uma variável do backend. O campo de foto exibe a imagem atual do aluno (caso exista) e permite ao usuário enviar uma nova foto, caso desejado.

Além disso, há um script que, ao selecionar uma nova imagem no campo de upload, atualiza a visualização da foto no formulário em tempo real. Ou seja, quando o usuário escolhe uma nova imagem, ela é carregada automaticamente no local da foto atual, permitindo uma prévia da imagem antes do envio. O formulário inclui validações, como campos obrigatórios para informações essenciais (nome, telefone, escola, endereço), e um botão de envio para salvar as alterações feitas nos dados do aluno.

```
<main>
        <form class="container-form" action="/editar-aluno/{{
aluno['id_aluno'] }}" method="post" enctype="multipart/form-data">
            <div class="itens-form">
                <section class="lado-esquerdo-container">
                    <label for="nomeAluno">Nome:</label>
                    <input type="text" id="nomeAluno" placeholder="ex:
João da Silva" name="nome-aluno" value="{{ aluno['nome_aluno'] }}"
required>
```

```
                    <label for="nomeResponsavel">Nome do
responsável:</label>
                    <input type="text" id="nomeResponsavel"
placeholder="ex: Maria de Souza" name="nome-responsavel" value="{{
aluno['nome_responsavel'] }}" required>

                    <label for="telefoneResponsavel">Telefone do
responsável:</label>
                    <input type="tel" maxlength="15"
onkeyup="handlePhone(event)" id="telefoneResponsavel" placeholder="ex:
(00) 0000-0000" name="telefone-responsavel" value="{{
aluno['telefone_responsavel'] }}" required>

                    <label for="escola">Escola:</label>
                    <input type="text" id="escola" placeholder="ex: Escola
Horizonte Azul" name="escola" value="{{ aluno['escola'] }}" required>


                    <label for="emailAluno">Email Responsavel:</label>
                    <input type="email" id="emailAluno" placeholder="ex:
nome@exemplo.com" name="email-aluno" value="{{ aluno['email_responsavel']
}}">
                </section>

                <section class="lado-direito-container">
                    <label for="condicaoMedica">Observações Aluno:</label>
                    <input type="text" id="condicaoMedica"
placeholder="Insira um texto sobre, se não tem escreva 'Não tem'"
name="condicao-medica" value="{{ aluno['condicao_medica'] }}">

                    <label for="enderecoAluno">Endereço:</label>
```

```
                    <input type="text" id="enderecoAluno" placeholder="ex:
Rua das Flores, 123 - Bairro, Cidade, Estado" name="endereco-aluno"
value="{{ aluno['endereco'] }}" required>

                    <!-- Exibir a foto atual do aluno -->
                    <label for="fotoAluno">Foto Atual:</label>
                    <img src="{{ aluno['foto_aluno'] or
'/static/images/placeholder.png' }}" alt="Foto do Aluno" style="max-width:
100px;">

                    <!-- Campo para o usuário fazer o upload de uma nova
foto -->
                    <label for="fotoAluno">Alterar Foto do Aluno:</label>
                    <input type="file" id="fotoAluno" accept="image/*"
name="foto-aluno">

                    <div class="btn-container">
                        <button type="submit" class="cadastrar">Salvar
Alterações</button>
                    </div>
                </section>
            </div>
        </form>
        <script>

document.getElementById('fotoAluno').addEventListener('change',
function(event) {
                const [file] = event.target.files;
                if (file) {
                    const img = document.querySelector('img[alt="Foto do
Aluno"]');
                    img.src = URL.createObjectURL(file);
                }
```

```
            });
        </script>
    </main>
```

2-Python

Essa rota Flask (`/editar-aluno/<int:id_aluno>`) é responsável por permitir a edição dos dados de um aluno no sistema. Quando a requisição é do tipo `GET`, o código busca as informações do aluno no banco de dados com base no `id_aluno` fornecido na URL. Se o aluno for encontrado, os dados são passados para o template `editar-aluno.html` para preencher o formulário de edição. Caso o aluno não seja encontrado, uma mensagem de erro 404 é retornada. Isso permite que o usuário veja e edite os dados do aluno diretamente no formulário.

Quando o formulário é submetido via `POST`, o código coleta os dados inseridos, incluindo o nome, telefone, escola, endereço, e-mail, condição médica e foto do aluno. Se uma nova foto for carregada, o sistema verifica o formato da imagem (somente `.png`, `.jpg`, ou `.jpeg`) e a salva no diretório apropriado. Caso contrário, a foto existente é mantida. Após atualizar as informações no banco de dados, o código confirma a transação com `conn.commit()` e redireciona o usuário para a página `/listar-alunos`. Isso garante que as alterações sejam salvas e refletidas na lista de alunos do sistema.

```
@app.route('/editar-aluno/<int:id_aluno>', methods=['GET', 'POST'])
def editar_aluno(id_aluno):
    conn = Conexao.conectar()
    cursor = conn.cursor(dictionary=True)

    # Quando o método for GET, busque os dados do aluno para preencher o
formulário
    if request.method == 'GET':
        cursor.execute("SELECT * FROM tb_alunos WHERE id_aluno = %s",
(id_aluno,))
        aluno = cursor.fetchone()
```

```
        if aluno:
            return render_template('editar-aluno.html', aluno=aluno)
        else:
            return "Aluno não encontrado", 404

    # Quando o método for POST, salve os dados atualizados
    elif request.method == 'POST':
        nome_aluno = request.form.get('nome-aluno')
        condicao_medica = request.form.get('condicao-medica')
        escola = request.form.get('escola')
        nome_responsavel = request.form.get('nome-responsavel')
        endereco = request.form.get('endereco-aluno')
        telefone_responsavel = request.form.get('telefone-responsavel')
        email_responsavel = request.form.get('email-aluno')

        # Verificar se o usuário fez upload de uma nova foto
        foto_aluno = request.files.get('foto-aluno')
        if foto_aluno and foto_aluno.filename != '':
            # Validar tipo de arquivo de imagem
            if foto_aluno.filename.lower().endswith(('.png', '.jpg',
'.jpeg')):
                foto_path = f"/static/images/{foto_aluno.filename}"
                foto_aluno.save(os.path.join('static', 'images',
foto_aluno.filename))
            else:
                return "Formato de imagem inválido. Use PNG, JPG ou
JPEG.", 400
        else:
            # Manter a foto existente
            cursor.execute("SELECT foto_aluno FROM tb_alunos WHERE
id_aluno = %s", (id_aluno,))
            foto_path = cursor.fetchone()['foto_aluno']
```

```
        # Atualizar os dados do aluno no banco de dados
        cursor.execute("""
            UPDATE tb_alunos
            SET nome_aluno = %s, condicao_medica = %s, escola = %s,
nome_responsavel = %s, endereco = %s, telefone_responsavel = %s,
email_responsavel = %s, foto_aluno = %s
            WHERE id_aluno = %s
        """, (nome_aluno, condicao_medica, escola, nome_responsavel,
endereco, telefone_responsavel, email_responsavel, foto_path, id_aluno))

        conn.commit()
        conn.close()

        return redirect('/listar-alunos')
const fotoAlunoInput = document.getElementById('fotoAluno');
const fotoAlunoPreview = document.getElementById('foto-aluno-preview');

fotoAlunoInput.addEventListener('change', (e) => {
  const file = e.target.files[0];
  const reader = new FileReader();

  reader.onload = (e) => {
    fotoAlunoPreview.src = e.target.result;
  };

  reader.readAsDataURL(file);
});
```

APÊNDICE H: RF008– Quebra de contato

1-HTML

Este código HTML apresenta uma interface para a funcionalidade de quebra de contrato entre um motorista e um aluno. A estrutura é organizada em seções que alertam o motorista sobre as implicações da rescisão, destacando pontos como impacto na reputação, necessidade de comunicação clara e revisão das cláusulas contratuais. Utiliza listas para facilitar a leitura e compreensão dos tópicos importantes. Além disso, oferece uma mensagem final que questiona a decisão do motorista, acompanhada de um botão de confirmação. Este design visa promover uma reflexão cuidadosa antes da ação definitiva, assegurando que o motorista esteja ciente das consequências.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Bebas+Neue&display=swap"
rel="stylesheet">

    <link rel="stylesheet" href="/static/styles/quebra-contrato.css">
    <link rel="stylesheet" href="/static/styles/variaveis.css">
    <link rel="stylesheet" href="/static/styles/style.css">

    <title>Quebra de Contrato</title>
</head>
<body>
    <nav>
        <a href="/listar-alunos"><button class="botao">Voltar</button></a>

    </nav>
```

```
    <div class="container">
        <h1>Quebra de Contrato</h1>

        <h2>Alerta ao Motorista</h2>
        <p>Prezado Motorista,</p>

        <p>Você está prestes a encerrar o contrato com o aluno. </p>

        <br>
        <p>Antes de prosseguir, considere os seguintes pontos importantes:</p>

        <ul>
            <li><strong>Impacto da Decisão:</strong> A rescisão pode afetar
tanto sua reputação profissional quanto a do aluno. Avalie se essa é a melhor
escolha.</li><br>
            <li><strong>Comunicação Clara:</strong> É essencial informar o
aluno sobre sua decisão de maneira respeitosa e transparente. Uma conversa
aberta pode evitar mal-entendidos e garantir que ambos compreendam as
razões.</li><br>
            <li><strong>Revisão do Contrato:</strong> Verifique as cláusulas
do contrato original para entender possíveis penalidades ou obrigações. Isso
garantirá que você esteja ciente de suas responsabilidades legais.</li><br>
            <li><strong>Alternativas:</strong> Considere se há maneiras de
resolver os problemas atuais antes de encerrar o contrato. Às vezes, ajustes
simples podem melhorar a situação.</li><br>
            <li><strong>Consequências Futuras:</strong> Pense nas repercussões
a longo prazo de sua decisão. Como isso pode impactar suas futuras relações
com alunos e outros motoristas?</li><br>
        </ul>

        <p>Reflita cuidadosamente sobre sua decisão e, se necessário, busque
aconselhamento antes de prosseguir. Sua escolha é importante e pode definir
sua trajetória profissional.</p>

        <p>Atenciosamente, MoveBR</p>

        <a href="/excluir-aluno/{{id_aluno}}"><button>Confirmar</button></a>
    </div>
</body>
</html>
```

2-CSS

Este CSS define o estilo visual de uma página web, utilizando uma paleta de cores e tipografia que criam uma aparência moderna e limpa. O `body` configura a

fonte e o fundo da página, enquanto o `header` utiliza um gradiente suave e uma sombra sutil para dar profundidade. A classe `.container` estiliza uma área centralizada, com fundo branco, padding e bordas arredondadas, proporcionando um foco ao conteúdo. As regras para os elementos `h1` e `p` definem cores e fontes, garantindo boa legibilidade. Finalmente, o botão é estilizado com uma cor de fundo chamativa e efeitos de hover, melhorando a interatividade do usuário.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;

}
header {
    background-image: linear-gradient(to right, var(--cor-05) 0%, var(--cor-06) 100%); /* Gradiente suave de azul claro para azul petróleo */
    color: var(--cor-03); /* Texto em bege claro para contraste elegante */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15); /* Sombra suave para profundidade */
}
.logo img {
    width: 80px; /* Tamanho da imagem do logo */
    height: auto; /* Manter a proporção da imagem */
    margin-left: 10px;
}

.container {

    background: white;
    padding: 50px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    max-width: 550px;
    margin: auto;
    text-align: center;
}

h1 {
    color: #333;
}

p {
    color: #000000;
```

```
    font-family: "Bebas Neue", sans-serif;
    font-weight: 400;
    font-style: normal;
}



button {
    background-color: #d9534f;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
}

button:hover {
    background-color: #c9302c;
}
```

3-Banco de dados

A criação da tabela **`contratos_fechados`** visa registrar informações sobre contratos firmados entre alunos e motoristas. A tabela possui um campo `id_contrato` que identifica unicamente cada contrato, além de `id_aluno` e `cpf_motorista` que estabelecem referências às tabelas **`tb_alunos`** e **`tb_motoristas`**, respectivamente. As chaves estrangeiras garantem a integridade referencial, assegurando que os registros de alunos e motoristas associados sejam válidos. Essa estrutura facilita o rastreamento de contratos e suas respectivas partes envolvidas.

```
-- Criação da tabela contratos_fechados
CREATE TABLE contratos_fechados (
    id_contrato INT AUTO_INCREMENT PRIMARY KEY,
    id_aluno INT,
    cpf_motorista int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES
tb_motoristas(cpf_motorista)
```

```
);
```

4-Python

```
@app.route("/quebra-contrato/<id_aluno>", methods=['GET'])
def quebra_foto(id_aluno):
    return render_template("quebra-contrato.html", id_aluno = id_aluno)
```

5- Javascript

O código JavaScript fornecido implementa interatividade em uma página web. Ao carregar o documento, ele adiciona um evento de clique ao ícone do menu, que alterna a classe "show" no menu, permitindo que ele apareça ou desapareça de maneira animada. Também inclui uma função para abrir ou fechar um segundo menu (com `menu1`), alterando seu estilo de exibição entre "grid" e "none" e ajustando o ícone do botão de alternância conforme necessário.

Além disso, o código seleciona todos os botões que representam meses e adiciona um evento de clique a cada um. Quando um botão é clicado, ele exibe uma seção de detalhes e atualiza o conteúdo para mostrar qual mês foi selecionado. Essa abordagem melhora a usabilidade da interface, permitindo que os usuários interajam de maneira dinâmica com a página.

```
document.addEventListener("DOMContentLoaded", function() {
    const menuIcon = document.getElementById('menu-icon');
    const menu = document.getElementById('menu');

    menuIcon.addEventListener('click', function() {
        // Adiciona ou remove a classe 'show' para mostrar/esconder
o menu
        if (menu.classList.contains('show')) {
            menu.classList.remove('show'); // Esconde o menu
(desliza para fora)
        } else {
            menu.classList.add('show'); // Mostra o menu (desliza
para dentro)
        }
    });
});
```

```
// Função para abrir e fechar o menu
document.getElementById('toggleBtn').addEventListener('click',
function() {
    var menu = document.getElementById('menu1');
    menu.style.display = menu.style.display === 'grid' ? 'none' :
'grid';
    this.innerHTML = menu.style.display === 'grid' ? '&#9660;' :
'&#9650;';
});

// Selecionar todos os botões de meses
const monthButtons = document.querySelectorAll('.month');
const details = document.getElementById('details');
const selectedMonth = document.getElementById('selectedMonth');

// Adicionar evento de clique para cada botão
monthButtons.forEach(button => {
    button.addEventListener('click', function() {
        // Exibir os detalhes
        details.style.display = 'block';
        // Atualizar o nome do mês nos detalhes
        selectedMonth.textContent = this.getAttribute('data-month');
    });
});
```

APÊNDICE L - Excluir Aluno

1-HTML

O trecho de código HTML exibe uma mensagem de confirmação para o encerramento do contrato de um aluno. A mensagem pergunta ao usuário se ele tem certeza de que deseja prosseguir. Um botão "Confirmar" é fornecido, que redireciona para a rota `/excluir-aluno/{{id_aluno}}`, onde `{{id_aluno}}` é substituído pelo ID real do aluno a ser excluído. Abaixo, uma segunda seção contém um botão "Excluir Aluno", que também redireciona para a rota `/quebra-contrato/{{aluno['id_aluno']}}`. Essas ações facilitam a exclusão do aluno, garantindo que o usuário esteja ciente da ação que está prestes a realizar.

```
<p>Você está prestes a encerrar seu contrato com o aluno. Tem certeza de que deseja
prosseguir?</p>
        <a href="/excluir-aluno/{{id_aluno}}"><button>Confirmar</button></a>
```

```
<div class="container-btn-excluir">
                        <a href="/quebra-contrato/{{aluno['id_aluno']}}"><button>Excluir
Aluno</button></a>
                    </div>
```

2-CSS

A classe CSS `.container-btn-excluir` aplica uma margem esquerda de 87% para posicionar o contêiner dos botões à direita da tela. Dentro deste contêiner, a classe `.container-btn-excluir button` estiliza os botões com um fundo colorido definido por `var(--cor-06)`, texto na cor `var(--cor-03)`, e remove a borda padrão. Adicionalmente, o botão tem um espaçamento interno, bordas arredondadas e um cursor de mão para indicar interatividade, com transições suaves para mudanças de cor e transformações ao passar o mouse.

```
.container-btn-excluir {
    margin-left: 87%;
}

.container-btn-excluir button {
    background-color: var(--cor-06); /* Cor de fundo */
    color: var(--cor-03); /* Cor do texto */
```

```
    border: none; /* Remove borda padrão */
    padding: 12px; /* Espaçamento interno */
    border-radius: 6px; /* Bordas arredondadas */
    cursor: pointer; /* Cursor de mão */
    transition: background-color 0.3s ease, transform 0.2s ease; /* Transições suaves */
}
```

3-Python

O código define um método `excluir_aluno` que remove um registro de aluno da tabela `tb_alunos` com base no `id_aluno` fornecido. Ele estabelece uma conexão com o banco de dados, executa uma instrução SQL de exclusão e, em seguida, confirma a transação antes de fechar a conexão. O endpoint Flask `/excluir-aluno/<id_aluno>` gerencia requisições `GET` e `POST`. Ao receber uma requisição `GET`, verifica se um usuário está logado na sessão; se sim, chama o método `excluir_aluno` para realizar a exclusão e redireciona o usuário para a página de listagem de alunos. Isso garante que apenas usuários autenticados possam realizar a ação de exclusão.

```
def excluir_aluno(self, id_aluno):

        mydb = Conexao.conectar()
        mycursor = mydb.cursor()

        sql = f"DELETE  FROM tb_alunos WHERE id_aluno = {id_aluno}"
        mycursor.execute(sql)
        mydb.commit()
        mydb.close()
        return True
```

```
@app.route("/excluir-aluno/<id_aluno>", methods=['GET', 'POST'])
def excluir_aluno(id_aluno):
    if request.method == 'GET':
        if 'usuario_logado' in session:
            usuario = Usuario()
            usuario.excluir_aluno(id_aluno)
            return redirect('/listar-alunos')
```

APÊNDICE H: Histórico de pagamento

1-HTML

Este código HTML é de uma página de histórico de pagamentos de alunos, com funcionalidades para listar, filtrar e editar/excluir registros. O cabeçalho contém um botão de retorno para a página inicial e o logo do sistema. A seção principal exibe um formulário para cadastrar novos pagamentos e outro para filtrar os pagamentos por mês. Se houver pagamentos registrados, eles são listados em uma tabela com as informações de nome do aluno, método de pagamento, data, valor, e opções de editar ou excluir. Para alunos com pagamentos pendentes, a tabela também mostra um status de "Pendente". Além disso, há um cálculo dinâmico do valor total de todos os pagamentos realizados. O código inclui também o uso de JavaScript para somar os valores dos pagamentos e exibi-los no final da página.

```
 <div class="container">
            <div class="cabecalho-container">
                <img src="/static/icones/dinheiro.png" alt="Ícone de dinheiro"
class="icone-dinheiro">
                <h2>Histórico de Pagamentos</h2>
            </div>
            <div class="container-historico">
                <form action="/gerar-pagamento" method="get" class="ger-
pagamento">
                    <div class="ger-pagamento-container">
                        <button type="submit" class="ger-pagamento">+
Cadastrar novo pagamento</button>
                    </div>
                </form>
                <form action="/historico_pagamento_filtro" method="post">
                    <div class="container-mes-pagamento">
                        <select name="mesPagamento">
                            <option value="" disabled selected>Todos</option>
                            <option value="janeiro">Janeiro</option>
                            <option value="fevereiro">Fevereiro</option>
                            <option value="março">Março</option>
                            <option value="abril">Abril</option>
                            <option value="maio">Maio</option>
                            <option value="junho">Junho</option>
                            <option value="julho">Julho</option>
                            <option value="agosto">Agosto</option>
                            <option value="setembro">Setembro</option>
```

```
                            <option value="outubro">Outubro</option>
                            <option value="novembro">Novembro</option>
                            <option value="dezembro">Dezembro</option>
                        </select>
                        <button type="submit">Pesquisar</button>
                    </div>
                </form>
            </div>

            <div class="informacoes">
                {% if pagamentos %}
                <table class="alunos-listados">
                    <thead>
                        <tr>
                            <th><center>Nome do Aluno</center></th>
                            <th><center>Método de Pagamento</center></th>
                            <th><center>Data do Pagamento</center></th>
                            <th><center>Valor do Pagamento</center></th>
                            <th><center>Ações</center></th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for pagamento in pagamentos %}
                        <tr>
                            <td><strong>{{ pagamento.nome_aluno
}}</strong></td>
                            <td>{{ pagamento.metodo_pagamento }}</td>
                            <td>{{ pagamento.data_pagamento }}</td>
                            <td><strong class="valor_pagamento">R$ {{
pagamento.valor_pagamento }}</strong></td>
                            <td class="td-buttons">
                                <form action="/editar-pagamento/{{
pagamento.id_pagamento }}" method="get">
                                    <button class="btn-editar"
type="submit"><img src="/static/icones/editar-icone.png" alt=""></button>
                                </form>
                                <form action="/excluir-historico/{{
pagamento.id_pagamento }}" method="post">
                                    <button class="btn-excluir"
type="submit"><img src="/static/icones/deletar-pagamento.png" alt=""></button>
                                </form>
                            </td>
                        </tr>
                        {% endfor %}
                        {% for pendentes in alunos_pendentes %}
                        <tr>
                            <td><strong>{{ pendentes.nome_aluno
}}</strong></td>
```

```
                            <td>Pendente</td>
                            <td>Pendente</td>
                            <td>Pendente</td>
                            <td class="td-buttons">
                                <form action="/excluir-historico/{{
pendentes.id_pagamento }}" method="post">
                                    <button class="btn-excluir"
type="submit"><img src="/static/icones/deletar-pagamento.png" alt=""></button>
                                </form>
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
                {% else %}
                <p>Nenhum pagamento encontrado para este aluno.</p>
                {% endif %}
            </div>

        <h2 class="valor">Valor Total: <span id="valor_total">R$
0,00</span></h2>
```

2-CSS

Este código CSS define o estilo de uma página com várias seções, incluindo um cabeçalho fixo, containers para exibição de informações de pagamento e uma tabela para listar alunos. O layout é responsivo, ajustando-se a diferentes tamanhos de tela, desde dispositivos móveis até telas grandes. O cabeçalho possui um fundo cinza-escuro com uma logo e botões de navegação. A tabela de pagamentos exibe informações de alunos, com colunas para editar ou excluir registros. Há também botões para gerar pagamentos e exibir valores totais. A paleta de cores é suave, utilizando tons de cinza, azul e laranja. O código também define estilos para tornar a interface mais interativa, como transições nos botões e efeitos de hover.

```
/* Importando a fonte */
@import url(https://fonts.googleapis.com/css?family=Roboto);

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Body */
body {
```

```
    font-family: 'Roboto', sans-serif;
    background-color: var(--cor-03); /* Off-white */
    color: var(--cor-01); /* Cinza-azulado escuro */
    display: flex;
    justify-content: center;
    align-items: flex-start; /* Ajuste para centralizar conteúdo */
    min-height: 100vh;
    padding: 20px;
}

/* Header */
header {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    height: 80px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    position: fixed;
    top: 0;
    width: 100%;
}

.button-back {
    background-color: transparent;
    border: none;
    transition: transform 0.3s ease;
    cursor: pointer;
}

.button-back img {
    width: 20px;
    height: auto;
}

.button-back:hover {
    transform: scale(1.1);
}

/* Logo */
.logo img {
    height: 30px;
}

/* Container principal */
.container {
    width: 80%;
```

```
    margin: 100px auto 20px; /* Adiciona margem superior para o header fixo */
    background-color: #fff; /* Branco */
    border-radius: 12px;
    box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.1);
}

/* Cabeçalho do histórico */
.cabecalho-container {
    background-color: var(--cor-05); /* Laranja */
    color: var(--cor-03); /* Branco */
    padding: 10px 20px;
    display: flex;
    align-items: center;
    width: 100%;
    border-radius: 12px 12px 0 0;
}

.icone-dinheiro {
    width: 30px;
    height: auto;
}

.cabecalho-container h2 {
    margin-left: 10px;
}
.container-historico{
    display: flex;
    justify-content: space-between;
    align-items: center;
}
/* Gerar pagamento */
.ger-pagamento-container {
    width: 100%;
    display: flex;
    justify-content: first baseline;
    padding: 20px;
}

/* Botão gerar pagamento */
.ger-pagamento button {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    border: none;
    padding: 8px 18px;
    border-radius: 4px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}
```

```
.ger-pagamento button:hover {
    background-color: var(--cor-02); /* Cinza-azulado claro */
}
.container-mes-pagamento{
    width: 100%;
    display: flex;
    justify-content: end;
    padding: 20px;
}
/* Estilo da tabela */
.alunos-listados {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

.alunos-listados thead {
    padding: 0;
}

.alunos-listados th, .alunos-listados td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #bdc3c7; /* Cor do fundo da tabela */
}

.alunos-listados th {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
}

.informacoes p {
    padding: 10px 0 10px 20px;
}

.valor {
    padding: 10px;
    width: 100%;
    display: flex;
    justify-content: end;
}

.valor h2 {
    color: var(--cor-01); /* Cinza-azulado escuro */
}

#valor_total {
    color: var(--cor-05); /* Laranja */
```

```
}

.nao-pago {
    display: flex;
}

.nao-pago h2 {
    color: red;
    font-weight: 700;
}
.btn-editar img{
    height: 30px;
    width: 30px;
}
.btn-editar{
    background-color: rgb(3, 136, 244);
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir{
    background-color: red;
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir img{
    height: 30px;
    width: 30px;
}
.td-buttons{
    display: flex;
    justify-content: space-around;
    align-items: center;
}
.th-pagamento{
    display: flex;
    justify-content: space-around;
    align-items: center;
    flex-direction: row;
}
/* Para telas menores que 576px (max-width: 575.98px) */
@media (max-width: 575.98px) {
    body {
        flex-direction: column;
        padding: 10px;
    }

    header {
```

```
        height: auto;
        flex-wrap: wrap;
        padding: 10px;
        text-align: center;
        width: 100%;
    }

    .logo img {
        height: 25px;
    }

    .container {
        width: 100%;
        margin: 80px 0 10px;
    }

    .ger-pagamento button {
        font-size: 12px;
        padding: 6px 12px;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 12px;
        padding: 8px;
    }

    .valor h2 {
        font-size: 18px;
    }
}

/* Para telas entre 576px e 767px (min-width: 576px e max-width: 767.98px) */
@media (min-width: 576px) and (max-width: 767.98px) {
    header {
        padding: 0 15px;
    }

    .container {
        width: 90%;
        margin: 90px auto 15px;
    }

    .ger-pagamento button {
        font-size: 13px;
        padding: 8px 14px;
    }

    .alunos-listados th,
```

```
    .alunos-listados td {
        font-size: 14px;
    }

    .valor h2 {
        font-size: 20px;
    }
}

/* Para telas entre 768px e 991px (min-width: 768px e max-width: 991.98px) */
@media (min-width: 768px) and (max-width: 991.98px) {
    header {
        padding: 0 20px;
    }

    .container {
        width: 85%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 15px;
    }

    .valor h2 {
        font-size: 22px;
    }
}

/* Para telas entre 992px e 1199px (min-width: 992px e max-width: 1199.98px)
*/
@media (min-width: 992px) and (max-width: 1199.98px) {
    .container {
        width: 75%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 16px;
    }
}

/* Para telas entre 1200px e 1399px (min-width: 1200px e max-width: 1399.98px)
*/
@media (min-width: 1200px) and (max-width: 1399.98px) {
    .container {
        width: 70%;
    }
```

```
    .alunos-listados th,
    .alunos-listados td {
        font-size: 17px;
    }
}

/* Para telas acima de 1400px (min-width: 1400px) */
@media (min-width: 1400px) {
    .container {
        width: 60%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 18px;
    }
}
```

3-Banco de dados

O banco de dados é composto por duas tabelas principais: **`historico_pagamentos`** e **`verificacao_pagamento`**. A tabela **`historico_pagamentos`** registra detalhes dos pagamentos dos alunos, incluindo informações como valor, data e referências a alunos e motoristas. Já a tabela **`verificacao_pagamento`** é responsável por verificar o estado dos pagamentos, associando responsáveis e alunos, e contém dados sobre o valor e o mês do pagamento.

Ambas as tabelas são projetadas para garantir a integridade referencial com as tabelas relacionadas, assegurando que as informações sejam consistentes e confiáveis ao longo do banco de dados. Essa estrutura permite um gerenciamento eficiente dos dados financeiros, facilitando a consulta e análise dos pagamentos realizados.

```
CREATE TABLE historico_pagamentos (
    id_pagamento INT AUTO_INCREMENT,
    valor_pagamento FLOAT,
    mes_pagamento INT,
    data_pagamento date,
    id_aluno INT,
    cpf_motorista INT,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
```

```
    FOREIGN KEY (cpf_motorista) REFERENCES
tb_motoristas(cpf_motorista),
    PRIMARY KEY (id_pagamento)
);

-- Criação da tabela tb_faltas
CREATE TABLE verificacao_pagamento(
    cpf_responsavel int,
    id_aluno INT,
    valor_pagamento int,
    mes_pagamento int,
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_responsavel) REFERENCES
tb_responsavel(cpf_responsavel)
);
```

4-Python

O código fornece diversas rotas para gerenciar o histórico de pagamentos, exclusão de alunos e visualização de contratos em uma aplicação Flask. A rota `/historico_pagamento` exibe todos os pagamentos registrados, enquanto a `/historico_pagamento_filtro` permite filtrar os pagamentos com base no mês e CPF do motorista logado. A rota `/listar_historico_aluno` mostra os pagamentos de um aluno específico. A rota `/gerar-pagamento` permite a criação de novos pagamentos, recuperando os dados do formulário. A rota `/excluir-historico` permite excluir pagamentos do histórico. Além disso, a `/listar-alunos` exibe todos os alunos com a opção de filtro, enquanto `/excluir-aluno` exclui um aluno do sistema. A rota `/quebra-contrato` exibe um template relacionado ao rompimento de contrato com o aluno.

```
@app.route("/historico_pagamento", methods=['GET'])
def historico_pagamento():
    pagamentos = Pagamentos()
    historico = pagamentos.listar_historico()  # Chama o método
listar_historico()

    # Renderiza o template com os pagamentos ou uma lista vazia
    return render_template("historico-pagamento.html", pagamentos=historico)
```

```
@app.route("/historico_pagamento_filtro", methods=['POST'])
def historico_pagamento_filtro():
    # Recupera o CPF do motorista logado a partir da sessão
    cpf_motorista = session.get("usuario_logado", {}).get("cpf")

    # Verifica se o motorista está logado
    if not cpf_motorista:
        return "Motorista não está logado", 401  # Retorna erro se não estiver
logado

    mes = request.form.get('mesPagamento')  # Usar 'get' para evitar KeyError

    pagamento = Pagamentos()

    # Se o mês foi fornecido, filtra os pagamentos e alunos pendentes por mês
    if mes:
        historico_pagamentos = pagamento.listar_historico_filtro(mes,
cpf_motorista)
        alunos_pendentes = pagamento.listar_alunos_pendentes(mes,
cpf_motorista)
    else:
        # Caso não tenha sido fornecido mês, busca todos os pagamentos
        historico_pagamentos = pagamento.listar_historico_filtro(None,
cpf_motorista)
        alunos_pendentes = pagamento.listar_alunos_pendentes(None,
cpf_motorista)

    return render_template("historico-pagamento.html",
                           pagamentos=historico_pagamentos,
                           alunos_pendentes=alunos_pendentes)



@app.route("/listar_historico_aluno/<int:id_aluno>", methods=['GET'])
def listar_historico_aluno(id_aluno):
    try:
        # Conectando ao banco de dados
        mydb = Conexao.conectar()
        mycursor = mydb.cursor()

        # Consulta SQL para listar os pagamentos do aluno específico
        sql = """
        SELECT nome_aluno, metodo_pagamento, data_pagamento, valor_pagamento,
id_pagamento
        FROM historico_pagamentos
        WHERE id_aluno = %s;
        """
        mycursor.execute(sql, (id_aluno,))
```

```
        resultados = mycursor.fetchall()

        # Organizando os dados em uma lista de dicionários
        pagamentos = [{
            "nome_aluno": linha[0],
            "metodo_pagamento": linha[1],
            "data_pagamento": linha[2],
            "valor_pagamento": linha[3],
            "id_pagamento": linha[4],
        } for linha in resultados]

        mydb.close()

        # Renderizando o template de histórico de pagamentos para o aluno
        return render_template("historico-pagamento.html",
pagamentos=pagamentos, id_aluno=id_aluno)

    except Exception as e:
        print(f"Erro ao listar pagamentos do aluno {id_aluno}: {e}")
        return "Erro ao listar pagamentos do aluno", 500

@app.route("/gerar-pagamento", methods=['GET', 'POST'])
def gerar_pagamento_get():
    if request.method == 'GET':
        usuario = Usuario()
        lista_alunos = usuario.listar_contratos_motorista()
        return render_template("gerar_pagamento.html", alunos=lista_alunos)
    else:
        # Pega os valores do formulário
        id_aluno = request.form.get("id_aluno")
        nome_aluno = request.form.get("id_aluno")
        data_pagamento = request.form.get("data_pagamento")
        mes_pagamento = request.form.get("mes_pagamento")
        valor_pagamento = float(request.form["valor_pagamento"])
        metodo_pagamento = request.form.get("metodo_pagamento")  # Novo campo
        cpf_motorista = session["usuario_logado"]["cpf"]

        # Instancia a classe Pagamentos e chama a função gerar_pagamento
        pagamento = Pagamentos()
        if pagamento.gerar_pagamento(id_aluno, nome_aluno, mes_pagamento,
data_pagamento, valor_pagamento, metodo_pagamento, cpf_motorista):
            return redirect("/historico_pagamento")
        else:
            return "Erro ao gerar o pagamento", 500



@app.route("/quebra-contrato/<id_aluno>", methods=['GET'])
def quebra_foto(id_aluno):
```

```
    return render_template("quebra-contrato.html", id_aluno=id_aluno)

@app.route("/excluir-aluno/<id_aluno>", methods=['GET', 'POST'])
def excluir_aluno(id_aluno):
    if request.method == 'GET':
        if 'usuario_logado' in session:
            usuario = Usuario()
            usuario.excluir_aluno(id_aluno)
            return redirect('/listar-alunos')




@app.route("/listar-alunos", methods=['GET', 'POST'])
def listar_alunos():
    usuario = Usuario()  # Inicializa a classe Usuario
    pesquisa = request.args.get("pesquisa-aluno")  # Obtém o parâmetro
'pesquisa-aluno' da URL

    if pesquisa:
        # Se uma pesquisa foi realizada, filtra os alunos pelo nome
        alunos_filtrados = usuario.listar_aluno_por_nome(pesquisa)  # Lista de
alunos filtrados pelo nome
    else:
        # Se não houver pesquisa, lista todos os alunos
        alunos_filtrados = usuario.listar_contratos_motorista()  # Lista de
todos os alunos

    # Independente de pesquisa, obtém a lista de todas as escolas disponíveis
    lista_completa_alunos = usuario.listar_contratos_motorista()  # Lista
completa de todos os alunos
    escolas = {aluno['escola'] for aluno in lista_completa_alunos}  # Conjunto
de todas as escolas (evita duplicatas)

    # Renderiza o template, passando os alunos filtrados e todas as escolas
    return render_template("listar-aluno.html", alunos=alunos_filtrados,
escolas=escolas)


@app.route("/excluir-historico/<int:id_pagamento>", methods=['POST'])
def excluir_historico(id_pagamento):
    if 'usuario_logado' in session:
        usuario = Pagamentos()
        if usuario.excluir_historico(id_pagamento):
            flash("Pagamento excluído com sucesso!")
        else:
            flash("Erro ao excluir pagamento.")
        return redirect(url_for('historico_pagamento'))  # Redireciona de
volta para a página de histórico de pagamentos.
```

```
    # Se o usuário não estiver logado, redireciona para o login
    flash("Você precisa estar logado para excluir pagamentos.")
    return redirect(url_for('/logar'))
```

O código define a classe `Pagamentos` com métodos para gerenciar o histórico de pagamentos de motoristas e alunos. O método `gerar_pagamento` insere um novo pagamento no banco de dados, verificando o nome do aluno se não for fornecido. O método `listar_historico` retorna todos os pagamentos registrados, associando-os aos alunos. O método `listar_historico_filtro` permite filtrar os pagamentos por mês e CPF do motorista. O método `listar_alunos_pendentes` recupera os alunos que não têm pagamento registrado para um mês específico. O método `excluir_historico` permite excluir um pagamento com base no ID. Todos os métodos utilizam transações com o banco de dados, tratando erros e realizando commits ou rollback conforme necessário.

```
from conexao import Conexao
class Pagamentos():
    def __init__(self):
        self.nome = None
        self.cpf = None
        self.email = None
        self.logado = False

    def gerar_pagamento(self, id_aluno, nome_aluno, mes, data, valor,
metodo_pagamento, cpf_motorista):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Se o nome_aluno não foi passado, consulte na tabela tb_alunos
            if not nome_aluno:
                # Consultar o nome do aluno com base no id_aluno
                sql_nome = "SELECT nome_aluno FROM tb_alunos WHERE id_aluno =
%s"
                mycursor.execute(sql_nome, (id_aluno,))
                result = mycursor.fetchone()

                if result:
                    nome_aluno = result[0]  # Pega o nome_aluno retornado da
consulta
                else:
```

```
                    print(f"Aluno com id {id_aluno} não encontrado.")
                    return False

            # Inserir os dados na tabela historico_pagamentos
            sql = """
            INSERT INTO historico_pagamentos
            (id_aluno, nome_aluno, data_pagamento, mes_pagamento,
valor_pagamento, metodo_pagamento, cpf_motorista)
            VALUES (%s, %s, %s, %s, %s, %s, %s);
            """
            mycursor.execute(sql, (id_aluno, nome_aluno, data, mes, valor,
metodo_pagamento, cpf_motorista))

            mydb.commit()
            mycursor.close()
            return True

        except Exception as e:
            mydb.rollback()
            print(f"Erro ao cadastrar pagamento: {e}")
            return False


    def listar_historico(self):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # SQL para buscar os pagamentos junto com o nome dos alunos e
método de pagamento
            sql = """
            SELECT tb_alunos.nome_aluno,
historico_pagamentos.metodo_pagamento, historico_pagamentos.data_pagamento,
                historico_pagamentos.valor_pagamento,
historico_pagamentos.id_aluno, historico_pagamentos.id_pagamento
            FROM historico_pagamentos
            INNER JOIN tb_alunos ON historico_pagamentos.id_aluno =
tb_alunos.id_aluno;
            """

            mycursor.execute(sql)
            resultados = mycursor.fetchall()
            historico = []

            # Iterando sobre os resultados e adicionando ao histórico
            for linha in resultados:
                historico.append({
                    "nome_aluno": linha[0],              # Nome do aluno (da
tabela tb_alunos)
```

```
                    "metodo_pagamento": linha[1],          # Método de
pagamento
                    "data_pagamento": linha[2],            # Data do pagamento
                    "valor_pagamento": linha[3],           # Valor do pagamento
                    "id_aluno": linha[4],                  # ID do aluno
                    "id_pagamento": linha[5],              # ID do pagamento
                })

            mydb.close()
            return historico

        except Exception as e:
            print(f"Erro ao listar histórico: {e}")
            return False

    def listar_historico_filtro(self, mes, cpf_motorista):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Se 'mes' for fornecido, filtra pelo mês e CPF do motorista
            if mes:
                sql = """
                SELECT historico_pagamentos.id_pagamento,
tb_alunos.nome_aluno, historico_pagamentos.metodo_pagamento,
                    historico_pagamentos.data_pagamento,
historico_pagamentos.valor_pagamento, historico_pagamentos.id_aluno
                FROM historico_pagamentos
                INNER JOIN tb_alunos ON historico_pagamentos.id_aluno =
tb_alunos.id_aluno
                WHERE historico_pagamentos.mes_pagamento = %s AND
historico_pagamentos.cpf_motorista = %s
                """
                mycursor.execute(sql, (mes, cpf_motorista))
            else:
                # Se 'mes' for None, seleciona todos os pagamentos do
motorista
                sql = """
                SELECT historico_pagamentos.id_pagamento,
tb_alunos.nome_aluno, historico_pagamentos.metodo_pagamento,
                    historico_pagamentos.data_pagamento,
historico_pagamentos.valor_pagamento, historico_pagamentos.id_aluno
                FROM historico_pagamentos
                INNER JOIN tb_alunos ON historico_pagamentos.id_aluno =
tb_alunos.id_aluno
                WHERE historico_pagamentos.cpf_motorista = %s
                """
                mycursor.execute(sql, (cpf_motorista,))
```

```
            resultados = mycursor.fetchall()
            historico = [{
                "id_pagamento": linha[0],
                "nome_aluno": linha[1],  # Nome do aluno vindo da tabela
tb_alunos
                "metodo_pagamento": linha[2],
                "data_pagamento": linha[3],
                "valor_pagamento": linha[4],
                "id_aluno": linha[5]
            } for linha in resultados]

            mydb.close()
            return historico

        except Exception as e:
            print(f"Erro ao listar histórico: {e}")
            return []

    def listar_alunos_pendentes(self, mes, cpf_motorista):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Verifica se o mês foi especificado
            if mes:
                sql = """
                SELECT a.nome_aluno, a.id_aluno
                FROM tb_alunos AS a
                INNER JOIN contratos_fechados AS c ON a.id_aluno = c.id_aluno
                LEFT JOIN historico_pagamentos AS hp
                ON a.id_aluno = hp.id_aluno AND hp.mes_pagamento = %s
                WHERE c.cpf_motorista = %s AND hp.id_pagamento IS NULL;
                """
                mycursor.execute(sql, (mes, cpf_motorista))
            else:
                # Caso o mês não seja especificado, retorna lista vazia
                return []

            resultados = mycursor.fetchall()
            alunos_pendentes = [{"nome_aluno": linha[0], "id_aluno": linha[1]}
for linha in resultados]

            mydb.close()
            return alunos_pendentes
        except Exception as e:
            print(f"Erro ao listar alunos pendentes: {e}")
            return []

    def excluir_historico(self, id_pagamento):
```

```
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Deletando o pagamento pelo ID correto
            sql = "DELETE FROM historico_pagamentos WHERE id_pagamento = %s"
            mycursor.execute(sql, (id_pagamento,))
            mydb.commit()
            mycursor.close()
            return True

        except Exception as e:
            print(f"Erro ao excluir pagamento: {e}")
            return False
```

APÊNDICE I: gerar pagamentos

1-HTML

O código HTML representa um formulário destinado à coleta de informações para gerar um pagamento. Utilizando a tag <form> com o método POST, os dados são enviados para o endpoint /gerar_pagamento. O formulário inclui um menu suspenso (<select>) para selecionar um aluno, além de campos de entrada (<input>) para nome do responsável, data de pagamento, valor e nome da escola.

Organizado em uma seção à esquerda (<section>), o formulário é apresentado de forma clara e intuitiva. Os dados são processados dinamicamente através de um template que itera sobre a lista de alunos. Um botão de envio (<button>) permite submeter o formulário, iniciando o processo de geração do pagamento e facilitando a coleta de dados para o sistema.

```
<form class="container-form" action="/gerar_pagamento" method="post">
            <div class="itens-form">
                <section class="lado-esquerdo-container">
                    <form action="/gerar_pagamento" method="post">
                        <label for="nomeAluno">Aluno pagante:</label>
                        <select id="nomeAluno" name="id_aluno">
                            <!-- Aqui a lista de alunos será preenchida
dinamicamente pelo backend -->
                            {% for aluno in alunos %}
                            <option value="{{ aluno.id }}">{{ aluno.nome
}}</option>
                            {% endfor %}
                        </select>

                        <label for="mesPagamento">Mês do pagamento:</label>
                        <input type="text" id="nomeResponsavel"
placeholder="ex: Maria de Souza" name="nomeResponsavel">

                        <label for="telefoneResponsavel">Data exata do
pagamento:</label>
                        <input type="text" id="telefoneResponsavel"
placeholder="ex: 17/10/2006" name="telefoneResponsavel">

                        <label for="valorPagamento">Valor do
pagamento:</label>
                        <input type="text" id="escola" placeholder="ex: Escola
Horizonte Azul" name="escola">
```

```
                        <button type="submit" class="botao">Gerar
pagamento</button></a></button>
                    </form>


                </section>


            </div>
        </form>
    </main>
```

3-Banco de dados

O código SQL define duas tabelas: `historico_pagamentos` e `verificacao_pagamento`. A tabela `historico_pagamentos` registra pagamentos com campos como `nome_aluno`, `valor_pagamento`, `mes_pagamento`, e `data_pagamento`, utilizando `id_pagamento` como chave primária e estabelecendo chaves estrangeiras para relacionar `id_aluno` e `cpf_motorista` com suas respectivas tabelas. A tabela `verificacao_pagamento` armazena informações de verificação de pagamentos, incluindo `estado_pagamento`, e também referencia `id_aluno` como chave estrangeira. Ambas as tabelas utilizam tipos de dados apropriados, como `VARCHAR`, `INT`, e `FLOAT`, para garantir a integridade e eficiência do armazenamento.

```
CREATE TABLE historico_pagamentos (
    nome_aluno VARCHAR(50),
    id_pagamento INT AUTO_INCREMENT PRIMARY KEY,
    valor_pagamento FLOAT,
    mes_pagamento INT,
    data_pagamento date,
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista),
    PRIMARY KEY (id_pagamento)
);


CREATE TABLE verificacao_pagamento(
    id_aluno INT,
    valor_pagamento int,
    mes_pagamento int,
```

```
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno)
);
```

4-Python

O código define uma classe `Usuario` que possui métodos para gerenciar pagamentos, incluindo a geração de registros na tabela `historico_pagamentos`. O método `gerar_pagamento` busca o nome do aluno através de um `JOIN` na tabela `tb_alunos` e insere um novo registro de pagamento. A aplicação utiliza Flask, com duas rotas para manipular as requisições: uma para exibir o formulário (`GET`) e outra para processar os dados enviados (`POST`). Na rota `POST`, verifica-se se o motorista está logado, recuperando seu ID da sessão. Se estiver, o pagamento é gerado e, em caso de sucesso, redireciona para a página de histórico de pagamentos; caso contrário, retorna um erro. A gestão de conexões e exceções é feita para garantir a integridade dos dados.

```
class Pagamentos():
    def __init__(self):
        self.nome = None
        self.cpf = None
        self.email = None
        self.logado = False

    def gerar_pagamento(self, id_aluno, mes, data, valor):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Primeiro, você faz um JOIN para buscar o nome do aluno na tabela tb_alunos
            sql_busca_nome = f"""
            SELECT nome_aluno
            FROM tb_alunos
            WHERE id_aluno = {id_aluno};
            """
            mycursor.execute(sql_busca_nome)
            nome_aluno = mycursor.fetchone()[0]  # Captura o nome do aluno

            # Agora, você insere na tabela historico_pagamentos com o nome do aluno
            sql = f"""
            INSERT INTO historico_pagamentos (id_aluno, nome_aluno, data_pagamento,
mes_pagamento, valor_pagamento)
```

```
            VALUES ('{id_aluno}', '{nome_aluno}', '{data}', '{mes}', '{valor}');
            """
            mycursor.execute(sql)

            mydb.commit()
            mycursor.close()

            return True

        except Exception as e:
            print(f"Erro ao cadastrar pagamento: {e}")
            return False
```

```
@app.route("/gerar_pagamento", methods=['GET'])
def gerar_pagamento_get():
    usuario = Usuario()
    lista_alunos = usuario.listar_aluno()
    return render_template("gerar_pagamento.html", alunos=lista_alunos)

@app.route("/gerar_pagamento", methods=['POST'])
def gerar_pagamento_post():
    id_aluno = request.values["id_aluno"]
    data = request.form["data"]
    mes = request.form["mes"]
    valor = request.form["valor"]

    # Recupera o id do motorista logado a partir da sessão
    id_motorista = session.get("id_motorista")

    # Verifica se o motorista está logado
    if not id_motorista:
        return "Motorista não está logado", 401  # Retorna erro se não estiver logado

    pagamento = Pagamentos()
    if pagamento.gerar_pagamento(id_aluno, data, mes, valor, id_motorista):
        return render_template("historico_pagamento.html")
    else:
        return "Erro ao gerar pagamento", 500
```

APENDICE G: RF007– Somar pagamentos já feitos

1-HTML

O elemento `<div>` com o ID **`details`** serve para exibir informações sobre os alunos que realizaram pagamentos em um mês específico, inicialmente oculto via CSS. O parágrafo interno fornece uma descrição contextual, enquanto a `<div>` com a classe **`btn-container`** contém um botão para somar os pagamentos, sugerindo uma interação do usuário. A estrutura utiliza propriedades de estilo para gerenciar a visibilidade e a apresentação do conteúdo.

```
  <h2 class="valor">Valor Total: <span id="valor_total">R$ 0,00</span></h2>
```

2-CSS

Este código CSS define o estilo de uma página com várias seções, incluindo um cabeçalho fixo, containers para exibição de informações de pagamento e uma tabela para listar alunos. O layout é responsivo, ajustando-se a diferentes tamanhos de tela, desde dispositivos móveis até telas grandes. O cabeçalho possui um fundo cinza-escuro com uma logo e botões de navegação. A tabela de pagamentos exibe informações de alunos, com colunas para editar ou excluir registros. Há também botões para gerar pagamentos e exibir valores totais. A paleta de cores é suave, utilizando tons de cinza, azul e laranja. O código também define estilos para tornar a interface mais interativa, como transições nos botões e efeitos de hover.

```
/* Importando a fonte */
@import url(https://fonts.googleapis.com/css?family=Roboto);

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Body */
body {
    font-family: 'Roboto', sans-serif;
    background-color: var(--cor-03); /* Off-white */
    color: var(--cor-01); /* Cinza-azulado escuro */
    display: flex;
    justify-content: center;
```

```
    align-items: flex-start; /* Ajuste para centralizar conteúdo */
    min-height: 100vh;
    padding: 20px;
}

/* Header */
header {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    height: 80px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    position: fixed;
    top: 0;
    width: 100%;
}

.button-back {
    background-color: transparent;
    border: none;
    transition: transform 0.3s ease;
    cursor: pointer;
}

.button-back img {
    width: 20px;
    height: auto;
}

.button-back:hover {
    transform: scale(1.1);
}

/* Logo */
.logo img {
    height: 30px;
}

/* Container principal */
.container {
    width: 80%;
    margin: 100px auto 20px; /* Adiciona margem superior para o header fixo */
    background-color: #fff; /* Branco */
    border-radius: 12px;
    box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.1);
}
```

```
/* Cabeçalho do histórico */
.cabecalho-container {
    background-color: var(--cor-05); /* Laranja */
    color: var(--cor-03); /* Branco */
    padding: 10px 20px;
    display: flex;
    align-items: center;
    width: 100%;
    border-radius: 12px 12px 0 0;
}

.icone-dinheiro {
    width: 30px;
    height: auto;
}

.cabecalho-container h2 {
    margin-left: 10px;
}
.container-historico{
    display: flex;
    justify-content: space-between;
    align-items: center;
}
/* Gerar pagamento */
.ger-pagamento-container {
    width: 100%;
    display: flex;
    justify-content: first baseline;
    padding: 20px;
}

/* Botão gerar pagamento */
.ger-pagamento button {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    border: none;
    padding: 8px 18px;
    border-radius: 4px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.ger-pagamento button:hover {
    background-color: var(--cor-02); /* Cinza-azulado claro */
}
.container-mes-pagamento{
```

```
    width: 100%;
    display: flex;
    justify-content: end;
    padding: 20px;
}
/* Estilo da tabela */
.alunos-listados {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

.alunos-listados thead {
    padding: 0;
}

.alunos-listados th, .alunos-listados td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #bdc3c7; /* Cor do fundo da tabela */
}

.alunos-listados th {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
}

.informacoes p {
    padding: 10px 0 10px 20px;
}

.valor {
    padding: 10px;
    width: 100%;
    display: flex;
    justify-content: end;
}

.valor h2 {
    color: var(--cor-01); /* Cinza-azulado escuro */
}

#valor_total {
    color: var(--cor-05); /* Laranja */
}

.nao-pago {
    display: flex;
}
```

```
.nao-pago h2 {
    color: red;
    font-weight: 700;
}
.btn-editar img{
    height: 30px;
    width: 30px;
}
.btn-editar{
    background-color: rgb(3, 136, 244);
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir{
    background-color: red;
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir img{
    height: 30px;
    width: 30px;
}
.td-buttons{
    display: flex;
    justify-content: space-around;
    align-items: center;
}
.th-pagamento{
    display: flex;
    justify-content: space-around;
    align-items: center;
    flex-direction: row;
}
/* Para telas menores que 576px (max-width: 575.98px) */
@media (max-width: 575.98px) {
    body {
        flex-direction: column;
        padding: 10px;
    }

    header {
        height: auto;
        flex-wrap: wrap;
        padding: 10px;
        text-align: center;
        width: 100%;
```

```
    }

    .logo img {
        height: 25px;
    }

    .container {
        width: 100%;
        margin: 80px 0 10px;
    }

    .ger-pagamento button {
        font-size: 12px;
        padding: 6px 12px;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 12px;
        padding: 8px;
    }

    .valor h2 {
        font-size: 18px;
    }
}

/* Para telas entre 576px e 767px (min-width: 576px e max-width: 767.98px) */
@media (min-width: 576px) and (max-width: 767.98px) {
    header {
        padding: 0 15px;
    }

    .container {
        width: 90%;
        margin: 90px auto 15px;
    }

    .ger-pagamento button {
        font-size: 13px;
        padding: 8px 14px;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 14px;
    }

    .valor h2 {
```

```
        font-size: 20px;
    }
}

/* Para telas entre 768px e 991px (min-width: 768px e max-width: 991.98px) */
@media (min-width: 768px) and (max-width: 991.98px) {
    header {
        padding: 0 20px;
    }

    .container {
        width: 85%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 15px;
    }

    .valor h2 {
        font-size: 22px;
    }
}

/* Para telas entre 992px e 1199px (min-width: 992px e max-width: 1199.98px)
*/
@media (min-width: 992px) and (max-width: 1199.98px) {
    .container {
        width: 75%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 16px;
    }
}

/* Para telas entre 1200px e 1399px (min-width: 1200px e max-width: 1399.98px)
*/
@media (min-width: 1200px) and (max-width: 1399.98px) {
    .container {
        width: 70%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 17px;
    }
}
```

```
/* Para telas acima de 1400px (min-width: 1400px) */
@media (min-width: 1400px) {
    .container {
        width: 60%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 18px;
    }
}
```

3-Bancos de dados

O banco de dados possui duas tabelas principais: **`historico_pagamentos`** e **`verificacao_pagamento`**, que são utilizadas para a funcionalidade de somar pagamentos. A tabela **`historico_pagamentos`** registra detalhes dos pagamentos dos alunos, incluindo valor, data e referências aos alunos e motoristas, permitindo um acompanhamento preciso das transações. Já a tabela **`verificacao_pagamento`** verifica o estado dos pagamentos, associando responsáveis a alunos e armazenando informações sobre o valor e o mês do pagamento, facilitando a soma e análise dos dados financeiros. Ambas as tabelas garantem a integridade referencial com as tabelas relacionadas, essencial para a precisão dos dados.

```
- Criação da tabela historico_pagamento
CREATE TABLE historico_pagamentos (
    id_pagamento INT AUTO_INCREMENT PRIMARY KEY,
    nome_aluno VARCHAR(50),
    valor_pagamento FLOAT,
    mes_pagamento VARCHAR(20),
    data_pagamento VARCHAR(10),
    metodo_pagamento VARCHAR(50),
    id_aluno INT,
    cpf_motorista VARCHAR(100),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);

-- Criação da tabela tb_faltas
CREATE TABLE verificacao_pagamento(
    id_aluno INT,
    valor_pagamento int,
```

```
    mes_pagamento int,
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno)
);
```

4-Python

A rota /gerar_pagamento é definida para processar requisições POST que visam gerar um pagamento. A função extrai dados do formulário, incluindo o ID do aluno, data, mês e valor do pagamento. Uma instância da classe Pagamentos é criada e o método gerar_pagamento é chamado com os dados obtidos. Se a operação for bem-sucedida, a função renderiza a página gerar_pagamento.html. Caso contrário, não há tratamento explícito para falhas, pois o retorno é apenas implícito.

```
@app.route("/gerar_pagamento", methods=['POST'])
def gerar_pagamento():
   id_aluno = request.form["id_aluno"]
   data = request.form["data"]
   mes = request.form["mes"]
   valor = request.form["valor"]
   pagamento = Pagamentos()
   if pagamento.gerar_pagamento(id_aluno, data, mes, valor):
      return render_template("gerar_pagamento.html")
```

A classe Pagamentos contém um método gerar_pagamento que insere um novo registro de pagamento no banco de dados. Primeiro, ela se conecta ao banco MySQL e executa uma consulta SQL para recuperar o nome do aluno baseado no ID fornecido. Após obter o nome, uma segunda consulta insere os dados do pagamento na tabela historico_pagamentos. O método utiliza commit para persistir as alterações e fecha o cursor. Em caso de erro, uma mensagem de erro é impressa no console, e a função retorna False.

```
class Pagamentos():
    def __init__(self):
        self.nome = None
        self.cpf = None
        self.email = None
        self.logado = False

    def gerar_pagamento(self, id_aluno, mes, data, valor):
        try:
            mydb = Conexao.conectar()
            mycursor = mydb.cursor()

            # Primeiro, você faz um JOIN para buscar o nome do aluno
na tabela tb_alunos
            sql_busca_nome = f"""
            SELECT nome_aluno
            FROM tb_alunos
            WHERE id_aluno = {id_aluno};
            """
            mycursor.execute(sql_busca_nome)
            nome_aluno = mycursor.fetchone()[0]  # Captura o nome do
aluno

            # Agora, você insere na tabela historico_pagamentos com
o nome do aluno
            sql = f"""
            INSERT INTO historico_pagamentos (id_aluno, nome_aluno,
data_pagamento, mes_pagamento, valor_pagamento)
            VALUES ('{id_aluno}', '{nome_aluno}', '{data}', '{mes}',
'{valor}');
            """
            mycursor.execute(sql)

            mydb.commit()
```

```
            mycursor.close()

            return True

        except Exception as e:
            print(f"Erro ao cadastrar pagamento: {e}")
            return False
```

APENDICE – Editar pagamento

1- HTML

O código HTML exibe um formulário de cadastro de pagamentos, permitindo que o usuário edite o pagamento de um aluno. A estrutura inclui campos como "Nome do Aluno", "Mês de Pagamento", "Data", "Valor" e "Método de Pagamento". O campo "Mês" é preenchido com opções de meses do ano, e o campo "Método de Pagamento" oferece opções como Cartão de Crédito, Cartão de Débito, PIX e Dinheiro, com a opção selecionada predefinida de acordo com o valor anterior. O valor do pagamento, data e nome do aluno são preenchidos automaticamente com os dados do pagamento existente. O formulário envia os dados para a rota "/editar-pagamento/{{ pagamento.id_pagamento }}" para ser processado via método POST.

```
        <main>
            <div class="cabecalho-container">
                <h2>Cadastro de Pagamentos</h2>
            </div>

            <form class="container-form" action="/editar-pagamento/{{
pagamento.id_pagamento }}" method="post">
                <div class="itens-form">
                    <section class="lado-esquerdo-container">

                        <label for="nomeAluno">Pagamento:</label>
                        <input id="nomeAluno" name="nomeAluno" value="{{
pagamento.nome_aluno }}">

                        </input>

                        <label for="mesPagamento">Mês:</label>
                        <select id="mesPagamento" name="mes_pagamento">
                            <option value="Janeiro" {% if
pagamento.mes_pagamento == 'Janeiro' %} selected {% endif
%}>Janeiro</option>
                            <option value="Fevereiro"{% if
pagamento.mes_pagamento == 'Fevereiro' %} selected {% endif
%}>Fevereiro</option>
```

```
                            <option value="Março"{% if
pagamento.mes_pagamento == 'Março' %} selected {% endif %}>Março</option>
                            <option value="Abril"{% if
pagamento.mes_pagamento == 'Abril' %} selected {% endif %}>Abril</option>
                            <option value="Maio"{% if
pagamento.mes_pagamento == 'Maio' %} selected {% endif %}>Maio</option>
                            <option value="Junho"{% if
pagamento.mes_pagamento == 'Junho' %} selected {% endif %}>Junho</option>
                            <option value="Julho"{% if
pagamento.mes_pagamento == 'Julho' %} selected {% endif %}>Julho</option>
                            <option value="Agosto"{% if
pagamento.mes_pagamento == 'Agosto' %} selected {% endif %}>Agosto</option>
                            <option value="Setembro"{% if
pagamento.mes_pagamento == 'Setembro' %} selected {% endif
%}>Setembro</option>
                            <option value="Outubro"{% if
pagamento.mes_pagamento == 'Outubro' %} selected {% endif
%}>Outubro</option>
                            <option value="Novembro"{% if
pagamento.mes_pagamento == 'Novembro' %} selected {% endif
%}>Novembro</option>
                            <option value="Dezembro"{% if
pagamento.mes_pagamento == 'Dezembro' %} selected {% endif
%}>Dezembro</option>
                        </select>

                        <label for="dataPagamento">Data:</label>
                        <input type="date" id="dataPagamento" value="{{
pagamento.data_pagamento }}" name="data_pagamento">

                        <label for="valorPagamento">Valor:</label>
                        <input type="text" id="valorPagamento" value="{{
pagamento.valor_pagamento }}" name="valor_pagamento">

                        <!-- Novo campo: Método de pagamento -->
                        <label for="metodoPagamento">Método de
Pagamento:</label>
                        <select id="metodoPagamento"
name="metodo_pagamento">
                            <option
value="pagamento['metodo_pagamento']">{{ pagamento.metodo_pagamento
}}</option>
                            <option value="Cartão de Crédito">Cartão de
Crédito</option>
                            <option value="Cartão de Débito">Cartão de
Débito</option>
                            <option value="PIX">PIX</option>
                            <option value="Dinheiro">Dinheiro</option>
                        </select>
```

```
                        <button type="submit" class="botao">Cadastrar
pagamento</button>
                    </section>
                </div>
            </form>

        </main>
```

2- CSS

Este código CSS define o estilo de uma página com várias seções, incluindo um cabeçalho fixo, containers para exibição de informações de pagamento e uma tabela para listar alunos. O layout é responsivo, ajustando-se a diferentes tamanhos de tela, desde dispositivos móveis até telas grandes. O cabeçalho possui um fundo cinza-escuro com uma logo e botões de navegação. A tabela de pagamentos exibe informações de alunos, com colunas para editar ou excluir registros. Há também botões para gerar pagamentos e exibir valores totais. A paleta de cores é suave, utilizando tons de cinza, azul e laranja. O código também define estilos para tornar a interface mais interativa, como transições nos botões e efeitos de hover.

```
/* Importando a fonte */
@import url(https://fonts.googleapis.com/css?family=Roboto);

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Body */
body {
    font-family: 'Roboto', sans-serif;
    background-color: var(--cor-03); /* Off-white */
    color: var(--cor-01); /* Cinza-azulado escuro */
    display: flex;
    justify-content: center;
    align-items: flex-start; /* Ajuste para centralizar conteúdo */
    min-height: 100vh;
    padding: 20px;
}

/* Header */
header {
```

```
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    height: 80px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    position: fixed;
    top: 0;
    width: 100%;
}

.button-back {
    background-color: transparent;
    border: none;
    transition: transform 0.3s ease;
    cursor: pointer;
}

.button-back img {
    width: 20px;
    height: auto;
}

.button-back:hover {
    transform: scale(1.1);
}

/* Logo */
.logo img {
    height: 30px;
}

/* Container principal */
.container {
    width: 80%;
    margin: 100px auto 20px; /* Adiciona margem superior para o header fixo
*/
    background-color: #fff; /* Branco */
    border-radius: 12px;
    box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.1);
}

/* Cabeçalho do histórico */
.cabecalho-container {
    background-color: var(--cor-05); /* Laranja */
    color: var(--cor-03); /* Branco */
    padding: 10px 20px;
```

```
    display: flex;
    align-items: center;
    width: 100%;
    border-radius: 12px 12px 0 0;
}

.icone-dinheiro {
    width: 30px;
    height: auto;
}

.cabecalho-container h2 {
    margin-left: 10px;
}
.container-historico{
    display: flex;
    justify-content: space-between;
    align-items: center;
}
/* Gerar pagamento */
.ger-pagamento-container {
    width: 100%;
    display: flex;
    justify-content: first baseline;
    padding: 20px;
}

/* Botão gerar pagamento */
.ger-pagamento button {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
    border: none;
    padding: 8px 18px;
    border-radius: 4px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.ger-pagamento button:hover {
    background-color: var(--cor-02); /* Cinza-azulado claro */
}
.container-mes-pagamento{
    width: 100%;
    display: flex;
    justify-content: end;
    padding: 20px;
}
/* Estilo da tabela */
```

```
.alunos-listados {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}

.alunos-listados thead {
    padding: 0;
}

.alunos-listados th, .alunos-listados td {
    padding: 12px;
    text-align: left;
    border-bottom: 1px solid #bdc3c7; /* Cor do fundo da tabela */
}

.alunos-listados th {
    background-color: var(--cor-01); /* Cinza-azulado escuro */
    color: var(--cor-03); /* Branco */
}

.informacoes p {
    padding: 10px 0 10px 20px;
}

.valor {
    padding: 10px;
    width: 100%;
    display: flex;
    justify-content: end;
}

.valor h2 {
    color: var(--cor-01); /* Cinza-azulado escuro */
}

#valor_total {
    color: var(--cor-05); /* Laranja */
}

.nao-pago {
    display: flex;
}

.nao-pago h2 {
    color: red;
    font-weight: 700;
}
.btn-editar img{
```

```
    height: 30px;
    width: 30px;
}
.btn-editar{
    background-color: rgb(3, 136, 244);
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir{
    background-color: red;
    padding: 5px;
    border: 2px transparent;
    border-radius: 300px;
}
.btn-excluir img{
    height: 30px;
    width: 30px;
}
.td-buttons{
    display: flex;
    justify-content: space-around;
    align-items: center;
}
.th-pagamento{
    display: flex;
    justify-content: space-around;
    align-items: center;
    flex-direction: row;
}
/* Para telas menores que 576px (max-width: 575.98px) */
@media (max-width: 575.98px) {
    body {
        flex-direction: column;
        padding: 10px;
    }

    header {
        height: auto;
        flex-wrap: wrap;
        padding: 10px;
        text-align: center;
        width: 100%;
    }

    .logo img {
        height: 25px;
    }
```

```
    .container {
        width: 100%;
        margin: 80px 0 10px;
    }

    .ger-pagamento button {
        font-size: 12px;
        padding: 6px 12px;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 12px;
        padding: 8px;
    }

    .valor h2 {
        font-size: 18px;
    }
}

/* Para telas entre 576px e 767px (min-width: 576px e max-width: 767.98px)
*/
@media (min-width: 576px) and (max-width: 767.98px) {
    header {
        padding: 0 15px;
    }

    .container {
        width: 90%;
        margin: 90px auto 15px;
    }

    .ger-pagamento button {
        font-size: 13px;
        padding: 8px 14px;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 14px;
    }

    .valor h2 {
        font-size: 20px;
    }
}
```

```
/* Para telas entre 768px e 991px (min-width: 768px e max-width: 991.98px)
*/
@media (min-width: 768px) and (max-width: 991.98px) {
    header {
        padding: 0 20px;
    }

    .container {
        width: 85%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 15px;
    }

    .valor h2 {
        font-size: 22px;
    }
}

/* Para telas entre 992px e 1199px (min-width: 992px e max-width:
1199.98px) */
@media (min-width: 992px) and (max-width: 1199.98px) {
    .container {
        width: 75%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 16px;
    }
}

/* Para telas entre 1200px e 1399px (min-width: 1200px e max-width:
1399.98px) */
@media (min-width: 1200px) and (max-width: 1399.98px) {
    .container {
        width: 70%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 17px;
    }
}

/* Para telas acima de 1400px (min-width: 1400px) */
@media (min-width: 1400px) {
```

```
    .container {
        width: 60%;
    }

    .alunos-listados th,
    .alunos-listados td {
        font-size: 18px;
    }
}
```

3- BANCO DE DADOS

O banco de dados possui duas tabelas principais: **`historico_pagamentos`** e **`verificacao_pagamento`**, que são utilizadas para a funcionalidade de somar pagamentos. A tabela **`historico_pagamentos`** registra detalhes dos pagamentos dos alunos, incluindo valor, data e referências aos alunos e motoristas, permitindo um acompanhamento preciso das transações. Já a tabela **`verificacao_pagamento`** verifica o estado dos pagamentos, associando responsáveis a alunos e armazenando informações sobre o valor e o mês do pagamento, facilitando a soma e análise dos dados financeiros. Ambas as tabelas garantem a integridade referencial com as tabelas relacionadas, essencial para a precisão dos dados.

```
- Criação da tabela historico_pagamento
CREATE TABLE historico_pagamentos (
    id_pagamento INT AUTO_INCREMENT PRIMARY KEY,
    nome_aluno VARCHAR(50),
    valor_pagamento FLOAT,
    mes_pagamento VARCHAR(20),
    data_pagamento VARCHAR(10),
    metodo_pagamento VARCHAR(50),
    id_aluno INT,
    cpf_motorista VARCHAR(100),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);

-- Criação da tabela tb_faltas
CREATE TABLE verificacao_pagamento(
    id_aluno INT,
    valor_pagamento int,
```

```
    mes_pagamento int,
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno)
);
```

4- PYTHON

O código define duas rotas Flask para editar registros de alunos e pagamentos. Na rota `/editar-aluno/<int:id_aluno>`, ao receber uma requisição GET, busca-se os dados do aluno no banco de dados e preenche um formulário para edição. Se o método for POST, os dados atualizados são salvos, incluindo a possibilidade de enviar uma nova foto do aluno. A rota `/editar-pagamento/<int:id_pagamento>` funciona de forma semelhante: com GET, busca os dados do pagamento e exibe no formulário para edição; com POST, atualiza o pagamento no banco, com base nas informações fornecidas no formulário. Em ambos os casos, após a atualização, o usuário é redirecionado para as respectivas páginas de listagem.

```
@app.route('/editar-aluno/<int:id_aluno>', methods=['GET', 'POST'])
def editar_aluno(id_aluno):
    conn = Conexao.conectar()
    cursor = conn.cursor(dictionary=True)

    # Quando o método for GET, busque os dados do aluno para preencher o
formulário
    if request.method == 'GET':
        cursor.execute("SELECT * FROM tb_alunos WHERE id_aluno = %s",
(id_aluno,))
        aluno = cursor.fetchone()

        if aluno:
            return render_template('editar-aluno.html', aluno=aluno)
        else:
            return "Aluno não encontrado", 404

    # Quando o método for POST, salve os dados atualizados
    elif request.method == 'POST':
        nome_aluno = request.form.get('nome-aluno')
        condicao_medica = request.form.get('condicao-medica')
        escola = request.form.get('escola')
        nome_responsavel = request.form.get('nome-responsavel')
        endereco = request.form.get('endereco-aluno')
        telefone_responsavel = request.form.get('telefone-responsavel')
```

```
        email_responsavel = request.form.get('email-aluno')

        # Verificar se o usuário fez upload de uma nova foto
        foto_aluno = request.files.get('foto-aluno')
        if foto_aluno and foto_aluno.filename != '':
            # Validar tipo de arquivo de imagem
            if foto_aluno.filename.lower().endswith(('.png', '.jpg',
'.jpeg')):
                foto_path = f"/static/images/{foto_aluno.filename}"
                foto_aluno.save(os.path.join('static', 'images',
foto_aluno.filename))
            else:
                return "Formato de imagem inválido. Use PNG, JPG ou JPEG.",
400
        else:
            # Manter a foto existente
            cursor.execute("SELECT foto_aluno FROM tb_alunos WHERE id_aluno
= %s", (id_aluno,))
            foto_path = cursor.fetchone()['foto_aluno']

        # Atualizar os dados do aluno no banco de dados
        cursor.execute("""
            UPDATE tb_alunos
            SET nome_aluno = %s, condicao_medica = %s, escola = %s,
nome_responsavel = %s, endereco = %s, telefone_responsavel = %s,
email_responsavel = %s, foto_aluno = %s
            WHERE id_aluno = %s
        """, (nome_aluno, condicao_medica, escola, nome_responsavel,
endereco, telefone_responsavel, email_responsavel, foto_path, id_aluno))

        conn.commit()
        conn.close()

        return redirect('/listar-alunos')

@app.route('/editar-pagamento/<int:id_pagamento>', methods=['GET', 'POST'])
def editar_pagamento(id_pagamento):
    conn = Conexao.conectar()
    cursor = conn.cursor(dictionary=True)

    # Quando o método for GET, busque os dados do aluno para preencher o
formulário
    if request.method == 'GET':
        query = """
            SELECT
                hp.id_pagamento,
                hp.metodo_pagamento,
                hp.data_pagamento,
                hp.valor_pagamento,
```

```
                hp.id_aluno,
                ta.nome_aluno
            FROM
                historico_pagamentos hp
            JOIN
                tb_alunos ta
            ON
                hp.id_aluno = ta.id_aluno
            WHERE
                hp.id_pagamento = %s
        """
        cursor.execute(query, (id_pagamento,))
        linha = cursor.fetchone()

        if linha:
            historico = {
                "id_pagamento": linha["id_pagamento"],
                "nome_aluno": linha["nome_aluno"],  # Nome do aluno vindo
da tabela tb_alunos
                "metodo_pagamento": linha["metodo_pagamento"],
                "data_pagamento": linha["data_pagamento"],
                "valor_pagamento": linha["valor_pagamento"],
                "id_aluno": linha["id_aluno"]
            }
            return render_template('editar-pagamento.html',
pagamento=historico)
        else:
            return "Pagamento não encontrado", 404

    elif request.method == 'POST':
        nome_aluno = request.form.get('nomeAluno')
        mes_pagamento = request.form.get('mes_pagamento')
        data_pagamento = request.form.get('data_pagamento')
        metodo_pagamento = request.form.get('metodo_pagamento')
        valor_pagamento = request.form.get('valor_pagamento')

        cursor.execute("""
            UPDATE historico_pagamentos
            SET nome_aluno = %s, mes_pagamento = %s, data_pagamento = %s,
metodo_pagamento = %s, valor_pagamento = %s
            WHERE id_pagamento = %s
        """, (nome_aluno, mes_pagamento, data_pagamento, metodo_pagamento,
valor_pagamento, id_pagamento ))

        conn.commit()
        conn.close()
```

APÊNDICE M – Conexão no banco de dados

1-Python

O código define uma classe `Conexao` com um método estático `conectar()` que estabelece uma conexão com um banco de dados MySQL. Utiliza a biblioteca `mysql.connector` para conectar-se ao banco, especificando detalhes como host, porta, usuário, senha e nome do banco de dados. A conexão é feita com o host local (`127.0.0.1`) e a porta padrão do MySQL (`3306`). O método retorna um objeto de conexão (`mydb`), que pode ser utilizado para executar consultas e interagir com o banco de dados na aplicação.

```
import mysql.connector

class Conexao():
    def conectar():
        mydb = mysql.connector.connect(
            host="127.0.0.1",
            port=3306,
            user="movebr",
            password="123456789",
            database="movebr"
            )

        return mydb
```

2-Banco de dados

O banco de dados é armazenado em um diretório chamado `banco_de_dados`, com um arquivo de script denominado `bd_script`. O script contém comandos SQL para criar um banco de dados chamado `movebr` e define várias tabelas necessárias para a aplicação, como `tb_motoristas`, `tb_alunos`, e `historico_pagamentos`, cada uma com campos específicos e tipos de dados apropriados. As tabelas também implementam chaves primárias e estrangeiras para garantir a integridade referencial entre os dados. Além disso, o script cria um usuário (`movebr`) com privilégios totais no banco de dados, permitindo operações de

leitura, escrita e modificação. Essa estrutura é essencial para gerenciar informações sobre motoristas, alunos e pagamentos de forma organizada e eficiente.

```
CREATE DATABASE movebr;
USE movebr;



-- Criação da tabela tb_motoristas
CREATE TABLE tb_motoristas (
    nome_motorista varchar(60),
    cpf_motorista VARCHAR(100) PRIMARY KEY,
    cnh VARCHAR(100),
    cnpj VARCHAR(100),
    cidade_motorista VARCHAR(100),
    endereco_motorista VARCHAR(100),
    tel_motorista VARCHAR(15),
    email_motorista VARCHAR(50),
    senha_motorista varchar(50)
);

-- Criação da tabela tb_alunos
CREATE TABLE tb_alunos (
    id_aluno  int AUTO_INCREMENT,
    nome_aluno VARCHAR(100),
    foto_aluno VARCHAR(300),
    condicao_medica varchar(255),
    escola VARCHAR(100),
    nome_responsavel varchar(100),
    endereco VARCHAR(100),
    telefone_responsavel varchar(100),
    email_responsavel varchar(100),
    idade int,
    cpf_motorista varchar(100),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista),
    PRIMARY KEY (id_aluno)
);
-- Criação da tabela tb_alunos_registrados
CREATE TABLE tb_alunos_clientes (
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);

-- Criação da tabela historico_pagamento
CREATE TABLE historico_pagamentos (
```

```
    nome_aluno VARCHAR(50),
    id_pagamento INT AUTO_INCREMENT PRIMARY KEY,
    valor_pagamento FLOAT,
    mes_pagamento INT,
    data_pagamento date,
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)

);

-- Criação da tabela tb_faltas
CREATE TABLE verificacao_pagamento(
    id_aluno INT,
    valor_pagamento int,
    mes_pagamento int,
    data_pagamento date,
    estado_pagamento int,
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno)
);

-- Criação da tabela tb_alunos_registrados
CREATE TABLE contratos_fechados (
    id_contrato INT AUTO_INCREMENT PRIMARY KEY,
    id_aluno INT,
    cpf_motorista varchar(100),
    FOREIGN KEY (id_aluno) REFERENCES tb_alunos(id_aluno),
    FOREIGN KEY (cpf_motorista) REFERENCES tb_motoristas(cpf_motorista)
);

-- Criação do usuário e concessão de privilégios
CREATE USER 'movebr'@'%' IDENTIFIED BY '123456789';

GRANT ALL PRIVILEGES ON movebr.* TO 'movebr'@'%' WITH GRANT OPTION;
```

APENDICE P- CSS geral

O CSS fornece um reset básico para margens e preenchimentos, assegurando consistência entre navegadores. A classe `.global-header` é estilizada com um gradiente de fundo, cores definidas por variáveis CSS, e uma sombra para profundidade visual. A logo dentro do cabeçalho é responsiva, com tamanhos variados para diferentes larguras de tela, garantindo que a imagem se ajuste adequadamente. O rodapé `.global-footer` também utiliza um gradiente e é fixado na parte inferior da página, com ajustes de padding e tamanho da fonte baseados em media queries. As media queries garantem que tanto o cabeçalho quanto o rodapé se adaptem dinamicamente a uma variedade de dispositivos, desde telefones até grandes desktops.

```
/* Reset básico para garantir consistência entre navegadores */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Estilização do cabeçalho para uso global */
.global-header {
    background-image: linear-gradient(to right, var(--cor-05) 0%, var(-
-cor-06) 100%);
    color: var(--cor-03);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15);
}

.global-header .logo img {
    width: 80px;
    height: auto;
    margin-left: 10px;
}
```

```
/* Estilização do rodapé para uso global */
.global-footer {
    background-image: linear-gradient(to right, var(--cor-06) 0%, var(-
-cor-05) 100%);
    color: var(--cor-03);
    padding: 30px;
    text-align: center;
    position: fixed;
    bottom: 0;
    width: 100%;
    box-shadow: 0 -2px 8px rgba(0, 0, 0, 0.2);
}

/* MEDIA QUERIES */

/* Small devices (landscape phones, 576px and up) */
@media (min-width: 576px) {
    .global-header .logo img {
        width: 70px; /* Logo menor para telas pequenas */
    }

    .global-footer {
        padding: 20px;
        font-size: 14px;
    }
}

/* Medium devices (tablets, 768px and up) */
@media (min-width: 768px) {
    .global-header .logo img {
        width: 90px; /* Aumenta o logo em tablets */
    }

    .global-footer {
        padding: 25px;
        font-size: 16px;
    }
}

/* Large devices (desktops, 992px and up) */
@media (min-width: 992px) {
    .global-header .logo img {
        width: 100px; /* Aumenta o logo para desktops */
    }
```

```
    .global-footer {
        padding: 30px;
        font-size: 16px;
    }
}

/* X-Large devices (large desktops, 1200px and up) */
@media (min-width: 1200px) {
    .global-header .logo img {
        width: 120px; /* Aumenta ainda mais o logo */
    }

    .global-footer {
        padding: 35px;
        font-size: 18px;
    }
}

/* XX-Large devices (larger desktops, 1400px and up) */
@media (min-width: 1400px) {
    .global-header .logo img {
        width: 140px; /* Logo maior para telas grandes */
    }

    .global-footer {
        padding: 40px;
        font-size: 20px;
    }
}
```

APENDICE Q – Variaveis

O trecho de código define variáveis CSS na pseudo-classe `:root`, permitindo a reutilização de cores em todo o documento. As variáveis `--cor-01` a `--cor-06` são atribuídas a valores hexadecimais que representam diferentes tons de cinza e azul. Essa abordagem facilita a manutenção do estilo, pois, ao alterar o valor de uma variável, a mudança se reflete em todos os lugares onde a variável é utilizada. O uso de variáveis melhora a legibilidade e organização do código CSS, promovendo um design mais coeso. Comentários indicam que as variáveis podem ser facilmente referenciadas ao longo do CSS.

```css
:root{
    --cor-01: #333; /* cinza escura */
    --cor-02: #676767; /* cinza claro */
    --cor-03: #fff; /* branco */
    --cor-04: #f4f4f4; /* offwhite */
    --cor-05: #6DA6B1; /* "azul" claro */
    --cor-06: #346D7E; /* "azul" escuro */
}

/*
var(--cor-01);
var(--cor-02);
var(--cor-03);
var(--cor-04);
var(--cor-05);
var(--cor-06);
*/
```

APENDICE R- Importações

O código importa várias funcionalidades essenciais do Flask para criar uma aplicação web. O Flask é um framework que facilita o gerenciamento de rotas e respostas HTTP. Funções como render_template ajudam a combinar dados com templates HTML, enquanto redirect e url_for facilitam a navegação entre diferentes páginas. O objeto request permite acessar dados das requisições, e session mantém informações do usuário entre as interações. Além disso, flash envia mensagens temporárias e jsonify converte dados Python para JSON, útil para APIs. As classes Usuario e Pagamentos gerenciam lógica relacionada a usuários e operações financeiras, enquanto upload_file indica funcionalidades de upload de arquivos.

```
from flask import Flask, render_template, redirect, url_for, request,
session, flash, jsonify
from usuario import Usuario
from pagamentos import Pagamentos
from upload_file import upload_file
from conexao import Conexao
```

APÊNDICE N– Configuração do banco de dados no azure

O código define uma função `upload_file` para enviar arquivos para o Azure Blob Storage. Ele utiliza a biblioteca `azure.storage.blob` para se conectar ao serviço de armazenamento, usando uma string de conexão que inclui credenciais e configurações da conta. Ao receber um arquivo, a função verifica se ele possui um nome e, em caso afirmativo, gera um timestamp para renomeá-lo, garantindo que o arquivo seja único. Em seguida, utiliza um cliente de blob para fazer o upload do arquivo para um contêiner específico, permitindo a sobreposição de arquivos com o mesmo nome. Se o upload for bem-sucedido, a função retorna a URL do arquivo armazenado; caso contrário, retorna uma string vazia, indicando falha.

```
from azure.storage.blob import BlobServiceClient
from datetime import datetime

def upload_file (file) -> str:
    AZURE_CONNECTION_STRING =
"DefaultEndpointsProtocol=https;AccountName=movebr;AccountKey=zTfue7
Cp+hW60myep4Owt8+oTulr/4VVkCybbzXZIOZep0jaNv2nrtqccg3zd4es/0gs97UJJF
i1+AStz0ge3w==;EndpointSuffix=core.windows.net"

    CONTAINER_NAME = "fotos-movebr"
    LINK_AZURE = "https://movebr.blob.core.windows.net/"

    blob_service_client =
BlobServiceClient.from_connection_string(AZURE_CONNECTION_STRING)

    if file.filename == '':
        return ""

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    file.filename = f"{timestamp}_{file.filename}"

    # Upload para o Azure Blob Storage
    try:
        blob_client =
blob_service_client.get_blob_client(container=CONTAINER_NAME,
blob=file.filename)
```

```
        blob_client.upload_blob(file, overwrite=True)

        return f"{LINK_AZURE}{CONTAINER_NAME}/{file.filename}"
    except:
        return ""
```