PUC MINAS

ENGENHARIA DE SOFTWARE

Matrícula: 860085
Disciplina: Tecnologias para Análise e Desenvolvimento de Sistemas

Sumário

1 DESCRIÇÃO DO PROBLEMA

O presente trabalho consiste no desenvolvimento de um sistema completo para gerenciamento de locadora de veículos, com integração entre front-end e back-end. O sistema visa solucionar os problemas de controle de frota, gestão de clientes e aluguéis de uma locadora, proporcionando uma interface intuitiva e funcional para as operações do dia a dia.

1.1 Objetivos do Sistema

O sistema desenvolvido tem como principais objetivos:

- Gerenciar o cadastro completo de clientes com validações
- Controlar o inventário de veículos disponíveis
- Gerenciar fabricantes e categorias de veículos
- Registrar e acompanhar aluguéis ativos, finalizados e cancelados
- Fornecer consultas personalizadas e filtros avançados
- Calcular automaticamente valores de aluguéis baseados em diárias
- Integrar front-end React com API REST em .NET Core

1.2 Tecnologias Utilizadas

O projeto foi desenvolvido utilizando as seguintes tecnologias:

Back-end:

- .NET Core 9.0
- Entity Framework Core
- SQL Server
- ASP.NET Core Web API
- Swagger/OpenAPI para documentação

Front-end:

- React 18 com TypeScript
- Tailwind CSS v4
- shadcn/ui (componentes)
- Lucide React (ícones)
- Fetch API para requisições HTTP

2 MODELAGEM DO SISTEMA

2.1 Diagrama de Entidades

O sistema é composto por 5 entidades principais com seus relacionamentos:

Entidade	Atributos Principais
Cliente	IdCliente, Nome, CPF, Email, Telefone, DataNascimento
Fabricante	IdFabricante, Nome, PaisOrigem
CategoriaVeiculo	IdCategoria, Nome, Descricao, ValorDiariaBase
Veiculo	IdVeiculo, Modelo, Placa, Ano, Cor, Quilometragem, IdFabricante, IdCategoria, Disponivel
Aluguel	IdAluguel, DataRetirada, DataPrevistaDevolucao, DataDevolucaoReal, ValorDiaria, ValorTotal, StatusAluguel, IdCliente, IdVeiculo

Figura 1: Entidades do Sistema

2.2 Relacionamentos

- **Fabricante** → **Veiculo**: Um fabricante pode ter vários veículos (1:N)
- **CategoriaVeiculo** → **Veiculo**: Uma categoria pode ter vários veículos (1:N)
- **Cliente** → **Aluguel**: Um cliente pode ter vários aluguéis (1:N)
- **Veiculo** → **Aluguel**: Um veículo pode ter vários aluguéis (1:N)

3 ARQUITETURA DA APLICAÇÃO

3.1 Arquitetura Back-end

O back-end foi estruturado seguindo os princípios de API REST e arquitetura em camadas:

- **Controllers**: Responsáveis por receber requisições HTTP e retornar respostas
- **Models**: Entidades do domínio mapeadas para o banco de dados
- **DTOs**: Objetos de transferência de dados para comunicação com o front-end
- **Data**: Contexto do Entity Framework e configurações de banco
- **Migrations**: Versionamento do esquema do banco de dados

3.2 Arquitetura Front-end

O front-end foi desenvolvido com arquitetura baseada em componentes:

- **Components**: Componentes React reutilizáveis
- **Services**: Camada de comunicação com a API
- **Pages**: Páginas principais do sistema
- **UI Components**: Biblioteca de componentes base (shadcn/ui)

4 ROTAS DA API

4.1 Rotas de Clientes

```
GET    /api/Clientes          # Listar todos os clientes
GET    /api/Clientes/{id}     # Buscar cliente por ID
POST   /api/Clientes          # Criar novo cliente
PUT    /api/Clientes/{id}     # Atualizar cliente
DELETE /api/Clientes/{id}     # Excluir cliente
```

Listing 1: Endpoints de Clientes

4.2 Rotas de Veículos

```
GET    /api/Veiculos                                        # Listar
    todos
GET    /api/Veiculos/{id}                                   # Buscar
    por ID
POST   /api/Veiculos                                        # Criar
PUT    /api/Veiculos/{id}                                   # Atualizar
DELETE /api/Veiculos/{id}                                   # Excluir
GET    /api/Veiculos/disponiveis/categoria/{idCategoria}    # Filtro
    1
GET    /api/Veiculos/alugados/fabricante/{idFabricante}     # Filtro
    2
```

Listing 2: Endpoints de Veículos

4.3 Rotas de Aluguéis

```
GET    /api/Alugueis                              # Listar todos
GET    /api/Alugueis/{id}                         # Buscar por ID
POST   /api/Alugueis                              # Criar
PUT    /api/Alugueis/{id}                         # Atualizar
DELETE /api/Alugueis/{id}                         # Excluir
GET    /api/Alugueis/ativos/cliente/{idCliente}   # Filtro 1
GET    /api/Alugueis/periodo?inicio=X&fim=Y       # Filtro 2
```

Listing 3: Endpoints de Aluguéis

4.4 Rotas de Fabricantes e Categorias

```
GET    /api/Fabricantes           # Listar fabricantes
POST   /api/Fabricantes           # Criar fabricante
PUT    /api/Fabricantes/{id}      # Atualizar fabricante
DELETE /api/Fabricantes/{id}      # Excluir fabricante

GET    /api/CategoriasVeiculo     # Listar categorias
POST   /api/CategoriasVeiculo     # Criar categoria
PUT    /api/CategoriasVeiculo/{id}    # Atualizar
DELETE /api/CategoriasVeiculo/{id}    # Excluir
```

Listing 4: Endpoints Auxiliares

5 FUNCIONALIDADES IMPLEMENTADAS

5.1 Gestão de Clientes

A tela de clientes permite:

- Cadastro completo com validação de CPF e email
- Listagem com busca por nome, CPF ou email
- Edição de dados cadastrais
- Exclusão com confirmação
- Visualização de data de nascimento formatada

Filtros implementados:

1. Busca textual em múltiplos campos (nome, CPF, email)
2. Filtro em tempo real conforme digitação

5.2 Gestão de Veículos

A tela de veículos oferece:

- Cadastro vinculado a fabricante e categoria
- Controle de disponibilidade
- Registro de quilometragem
- Listagem com informações completas
- Busca avançada

Filtros implementados:

1. Busca por modelo, placa ou fabricante
2. Filtro por disponibilidade (Disponível/Indisponível)

5.3 Gestão de Aluguéis

A tela de aluguéis é a mais completa, com:

- Seleção de cliente e veículo disponível
- Cálculo automático de valor total baseado em diárias
- Controle de quilometragem inicial e final
- Gestão de status (Ativo, Finalizado, Cancelado)
- Registro de datas de retirada e devolução

Filtros implementados:

1. Busca por cliente ou veículo
2. Filtro por status do aluguel
3. **Filtro por período de datas** (consulta personalizada)

5.4 Gestão de Fabricantes

- Cadastro de fabricantes
- Registro de país de origem
- Listagem e edição

Filtros implementados:

1. Busca por nome do fabricante
2. Busca por país de origem

5.5 Gestão de Categorias

- Cadastro de categorias de veículos
- Definição de valor base da diária
- Descrição detalhada da categoria

Filtros implementados:

1. Busca por nome da categoria
2. Busca por descrição

6 CONSULTAS PERSONALIZADAS

6.1 Consulta 1: Veículos Disponíveis por Categoria

Esta consulta permite filtrar veículos que estão disponíveis para locação dentro de uma categoria específica.

Rota: `GET /api/Veiculos/disponiveis/categoria/{idCategoria}`

Código do Controller:

```
[HttpGet("disponiveis/categoria/{idCategoria}")]
public async Task<IActionResult> GetVeiculosDisponiveisPorCategoria(
  int idCategoria)
{
  var veiculos = await _context.Veiculos
      .Where(v => v.Disponivel && v.IdCategoria == idCategoria)
      .Include(v => v.CategoriaVeiculo)
      .Include(v => v.Fabricante)
      .Select(v => new VeiculoDto { /* ... */ })
      .ToListAsync();

  return Ok(veiculos);
}
```

Listing 5: Consulta de Veículos Disponíveis

Exemplo de uso:

- Buscar todos os veículos da categoria "Sedan"disponíveis
- Útil para mostrar opções ao cliente durante reserva

6.2 Consulta 2: Aluguéis por Período

Permite buscar todos os aluguéis realizados em um intervalo de datas específico.

Rota: `GET /api/Alugueis/periodo?inicio=YYYY-MM-DD&fim=YYYY-MM-DD`

Código do Controller:

```
[HttpGet("periodo")]
public async Task<IActionResult> GetAlugueisPorPeriodo(
  [FromQuery] DateTime inicio,
  [FromQuery] DateTime fim)
{
  var alugueis = await _context.Alugueis
      .Where(a => a.DataRetirada >= inicio &&
                  a.DataRetirada <= fim)
      .Include(a => a.Cliente)
      .Include(a => a.Veiculo)
      .ThenInclude(v => v.Fabricante)
      .Select(a => new AluguelDto { /* ... */ })
      .ToListAsync();

  return Ok(alugueis);
}
```

Listing 6: Consulta de Aluguéis por Período

Exemplo de uso:

- Relatório mensal de aluguéis
- Análise de faturamento por período
- Implementado no front-end com interface visual

6.3 Consulta 3: Aluguéis Ativos por Cliente

Busca todos os aluguéis ativos (sem data de devolução) de um cliente específico.

Rota: `GET /api/Alugueis/ativos/cliente/{idCliente}`

Código do Controller:

```
[HttpGet("ativos/cliente/{idCliente}")]
public async Task<IActionResult> GetAlugueisAtivosPorCliente(
  int idCliente)
{
  var alugueis = await _context.Alugueis
      .Where(a => a.IdCliente == idCliente &&
                  a.DataDevolucaoReal == null)
      .Include(a => a.Cliente)
      .Include(a => a.Veiculo)
      .ToListAsync();

  return Ok(alugueis);
```

```
}
```

Listing 7: Consulta de Aluguéis Ativos

Exemplo de uso:

- Verificar se cliente tem pendências
- Listar veículos atualmente com o cliente

7 INTERFACE DO USUÁRIO

7.1 Wireframes

Os wireframes foram desenvolvidos seguindo princípios de UX/UI modernas:

Link dos Wireframes: [Inserir link do Figma aqui]

Características dos wireframes:

- Design responsivo
- Navegação intuitiva via sidebar
- Tabelas com ações rápidas
- Modais para formulários
- Feedback visual de loading
- Mensagens de erro amigáveis

7.2 Componentes Desenvolvidos

Componentes de UI Base:

- Button (botões com variantes)
- Input (campos de texto)
- Select (seleção dropdown)
- Table (tabelas responsivas)
- Dialog (modais)
- AlertDialog (confirmações)
- Badge (etiquetas de status)
- Label (rótulos de formulário)

Páginas Principais:

- ClientesPage
- VeiculosPage
- FabricantesPage
- CategoriasPage
- AlugueisPage

8 INTEGRAÇÃO FRONT-END E BACK-END

8.1 Camada de Serviços

Foi implementada uma camada de serviços para abstrair as chamadas à API:

```
// src/services/api.ts
const API_BASE_URL = 'https://localhost:44385/api';

export const api = {
async get<T>(endpoint: string): Promise<T> {
  const response = await fetch(`${API_BASE_URL}${endpoint}`);
  if (!response.ok) throw new Error('Erro ao buscar');
  return response.json();
},

async post<T>(endpoint: string, data: unknown): Promise<T> {
  const response = await fetch(`${API_BASE_URL}${endpoint}`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data),
  });
  return response.json();
},
// ... put, delete
};
```

Listing 8: Serviço Base de API

8.2 Configuração CORS

O back-end foi configurado para aceitar requisições do front-end:

```
builder.Services.AddCors(options => {
  options.AddPolicy("AllowReactApp", policy => {
      policy.WithOrigins("http://localhost:5173")
            .AllowAnyHeader()
            .AllowAnyMethod();
  });
});

```

```
app.UseCors("AllowReactApp");
```

Listing 9: Configuração CORS

8.3 Fluxo de Dados

1. Usuário interage com componente React
2. Componente chama serviço correspondente
3. Serviço faz requisição HTTP para API
4. Controller processa e consulta banco via EF Core
5. Resposta retorna como DTO
6. Front-end atualiza estado e re-renderiza

9 VALIDAÇÕES E TRATAMENTO DE ERROS

9.1 Validações no Back-end

- Validação de campos obrigatórios via Data Annotations
- Verificação de existência de entidades relacionadas
- Validação de regras de negócio (ex: veículo disponível)

9.2 Tratamento de Erros no Front-end

- Try-catch em todas as operações assíncronas
- Exibição de mensagens de erro amigáveis
- Loading states durante requisições
- Validação de formulários antes do envio

10 FUNCIONALIDADES DESTACADAS

10.1 Cálculo Automático de Valores

O sistema calcula automaticamente o valor total do aluguel:

```
const calculateValorTotal = () => {
const dataRetirada = new Date(formData.dataRetirada);
const dataPrevista = new Date(formData.dataPrevistaDevolucao);
const dias = Math.ceil(
  (dataPrevista.getTime() - dataRetirada.getTime())
  / (1000 * 3600 * 24)
);
return dias > 0 ? dias * formData.valorDiaria : 0;
};
```

Listing 10: Cálculo de Valor Total

10.2 Filtros Dinâmicos

Todos os filtros funcionam em tempo real:

```
const filteredVeiculos = veiculos.filter((veiculo) => {
const matchesSearch =
  veiculo.modelo.toLowerCase().includes(searchTerm.toLowerCase())
     ||
  veiculo.placa.toLowerCase().includes(searchTerm.toLowerCase());

const matchesDisponibilidade =
  disponibilidadeFilter === 'todos' ||
  (disponibilidadeFilter === 'disponivel' && veiculo.disponivel);

return matchesSearch && matchesDisponibilidade;
});
```

Listing 11: Filtro em Tempo Real

10.3 Estados de Loading

Feedback visual durante operações:

- Spinner durante carregamento inicial
- Botões desabilitados durante requisições
- Mensagens de sucesso/erro

11 ESTRUTURA DO PROJETO

11.1 Estrutura Back-end

```
LocadoraDeVeiculos/
 Controllers/
    AlugueisController.cs
    CategoriasVeiculoController.cs
    ClientesController.cs
```

```
    FabricantesController.cs
    VeiculosController.cs
 Data/
    ApplicationDbContext.cs
 DTO's/
    AluguelDto.cs
    VeiculoDto.cs
 Migrations/
 Models/
    Aluguel.cs
    CategoriaVeiculo.cs
    Cliente.cs
    Fabricante.cs
    Veiculo.cs
 Program.cs
```

11.2 Estrutura Front-end

```
frontend/
 src/
    components/
       ui/
       AlugueisPage.tsx
       CategoriasPage.tsx
       ClientesPage.tsx
       FabricantesPage.tsx
       VeiculosPage.tsx
    services/
       api.ts
       alugueisService.ts
       categoriasService.ts
       clientesService.ts
       fabricantesService.ts
       veiculosService.ts
    App.tsx
    main.tsx
    index.css
 package.json
```

12 TESTES E VALIDAÇÃO

12.1 Testes Realizados

Testes de CRUD:

- Criação de todas as entidades

- Listagem e busca por ID
- Atualização de registros
- Exclusão com verificação de integridade

Testes de Integração:

- Comunicação front-end $\leftrightarrow$ back-end
- Persistência no banco de dados
- Relacionamentos entre entidades
- Consultas personalizadas

Testes de Interface:

- Responsividade em diferentes tamanhos de tela
- Validação de formulários
- Feedback visual de ações
- Navegação entre páginas

13 CONCLUSÃO

O sistema de locadora de veículos desenvolvido atende todos os requisitos propostos, oferecendo uma solução completa e integrada para gestão de locações. A arquitetura utilizada permite fácil manutenção e expansão futura.

13.1 Objetivos Alcançados

- Integração completa front-end e back-end
- CRUD completo para todas as entidades
- Filtros personalizados em múltiplos campos
- Consultas avançadas ao banco de dados
- Interface moderna e responsiva
- Validações e tratamento de erros
- Cálculos automáticos de valores
- Documentação completa da API

13.2 Possíveis Melhorias Futuras

- Implementação de autenticação e autorização
- Relatórios em PDF
- Dashboard com gráficos de estatísticas
- Sistema de notificações para devoluções
- Integração com gateway de pagamento
- Histórico de manutenções dos veículos
- Sistema de avaliação de clientes

14 REFERÊNCIAS

- Microsoft. *ASP.NET Core Documentation*. Disponível em: `https://docs.microsoft.com/aspnet/core`
- React. *React Documentation*. Disponível em: `https://react.dev`
- Tailwind CSS. *Documentation*. Disponível em: `https://tailwindcss.com/docs`
- Entity Framework Core. *Documentation*. Disponível em: `https://docs.microsoft.com/ef/core`
- shadcn/ui. *Component Library*. Disponível em: `https://ui.shadcn.com`

15 ANEXOS

15.1 Repositório GitHub

Link do repositório: https://github.com/MirellyAlvarenga/LocadoradeVeiculos

15.2 Link do Vídeo

Link do Vídeo: https://youtu.be/SO-qbQkFzew

15.3 Link dos Wireframes

Link do Figma: https://www.figma.com/design/lwkzqPI5Z9TkUNZXuaHlx0/AlugueldeVeiculos?node-id=0-1t=1FPfy4cm1R43705j-1