DOCUMENTAÇÃO TÉCNICA DO PROJETO – API MANUTENÇÃO DE ATIVOS

Projeto: Sistema de Manutenção de Caminhões
Curso: Ciência da Computação
Integrantes: Alex Silva, Bruna Ferreira, Guilherme Grassi e Gustavo Grassi

1. OBJETIVO DO PROJETO

O projeto consiste em uma API REST em C# com Entity Framework Core e banco SQLite, desenvolvida para gerenciar caminhões e ordens de serviço. A aplicação permite consultar, cadastrar e excluir dados diretamente do banco, seguindo boas práticas de programação.

2. ESTRUTURA DA SOLUÇÃO

2.1 Modelagem de Dados

A base de dados utiliza o Entity Framework Core com SQLite e possui três entidades principais: Caminhão e Ordem de Serviço. O banco inicia com 50 caminhões cadastrados, cada um com ID único

2.2 Integração da API com a Solução

A API foi organizada em controllers independentes para cada entidade, com rotas CRUD documentadas no Swagger. As respostas são em JSON, mantendo o back-end separado do front-end.

3. ENDPOINTS DA API

A tabela a seguir apresenta as principais rotas configuradas na API para cada entidade:

MÉTODO	ROTA	DESCRIÇÃO / EXEMPLO
GET	/api/caminhoes	Listar todos os caminhões (JSON)
GET	/api/caminhoes/{id}	Buscar caminhão por ID (ex: /api/caminhoes/1)
POST	/api/caminhoes	Cadastrar novo caminhão (body JSON)
DELETE	/api/caminhoes/{id}	Excluir caminhão por ID
GET	/api/ordens	Listar todas as ordens de serviço (JSON)
GET	/api/ordens/{id}	Buscar ordem por ID
POST	/api/ordens	Criar nova ordem (body JSON)
DELETE	/api/ordens/{id}	Excluir ordem por ID

4. ORGANIZAÇÃO DO CÓDIGO

O projeto foi organizado em camadas seguindo boas práticas de separação de responsabilidades**:** **Models** com as entidades, **Data** com o contexto do banco, **Controllers** com as rotas da API e **Program.cs** configurando o EF Core, SQLite e Swagger.

5. JUSTIFICATIVA TÉCNICA

O SQLite foi escolhido por ser leve e fácil de configurar, ideal para uso acadêmico. O Entity Framework Core facilita o acesso aos dados, e o padrão REST garante integração simples com outras aplicações. O Swagger fornece documentação e testes interativos da API.

6.IMPLEMENTAÇÃO DO CLIENTE WEB (FRONT-END)

A camada de apresentação do sistema, ou Cliente Web (Front-end), foi desenvolvida em HTML5, CSS3 e JavaScript, fundamentada no princípio de desacoplamento total da API (Back-end). A comunicação entre as duas camadas é estabelecida exclusivamente por requisições HTTP, com tráfego de dados no formato JSON, conforme exigido pelo padrão RESTful. Isso garante que o front-end seja um cliente leve e a API permaneça escalável e reutilizável por outras plataformas.

6.1 Estrutura e Estilização (HTML e CSS)

A estrutura visual da aplicação é baseada em HTML e estilizada por meio de arquivos CSS dedicados por funcionalidade (e.g., global.css, caminhoes.css, ordens.css).

- O CSS é crucial para traduzir o estado lógico dos dados da API em **feedback visual** para o usuário. Por exemplo, a utilização de classes como. badge-status com cores distintas permitem representar de forma imediata o status de um caminhão (Ativo, Manutenção) ou de uma Ordem de Serviço (Aberta, Concluída), facilitando a leitura do sistema.
- O design da interface prioriza a **densidade de dados** e a **usabilidade** em sistemas de gestão, adotando um padrão de "tabelas ultra-compactas" para otimizar a visualização e a navegação em grandes volumes de registros.

6.2 Lógica de Consumo da API (JavaScript)

Todo o fluxo de interação com a API é gerenciado pelo JavaScript, que utiliza a interface **Fetch API** para realizar as chamadas assíncronas. Os scripts são organizados de forma modular por entidade e propósito (e.g., caminhoes.js, ordens.js, cadastrar.js).

- **Operação de Leitura (GET):** O JavaScript executa requisições GET de forma assíncrona para obter listas completas de entidades (ex: /api/caminhoes, /api/ordens). No **Dashboard**, o código realiza chamadas a diferentes *endpoints* simultaneamente e utiliza a resposta JSON para calcular e exibir métricas importantes, como o número de caminhões Ativos e ordens Abertas.
- **Operação de Criação (POST):** A submissão de novos dados é realizada através do método POST. O script é responsável por coletar os dados do formulário, realizar o tratamento e a validação local (ex: conversão de valores monetários e datas) e os empacotar em um objeto JSON, que é enviado no corpo (body) da requisição para o *endpoint* de cadastro.
- **Operações de Manutenção (PUT e DELETE):** A edição e a exclusão de recursos (conforme os *endpoints* /api/{entidade}/{id}) são tratadas pelo JavaScript, que utiliza o **ID** da entidade, extraído dos parâmetros da URL, para montar a requisição.
- O método PUT é utilizado para **atualizar** o recurso, enviando o objeto completo atualizado para o *endpoint* específico.

- O método DELETE é acionado nas ações da tabela, enviando uma requisição simples ao *endpoint* de exclusão para remover o registro no banco de dados.
- A metodologia empregada no JavaScript garante que a **lógica de apresentação** esteja separada da **lógica de domínio** da API, finalizando a implementação da arquitetura Cliente-Servidor proposta.