Aqui está um Plano de Ação Detalhado para adaptar seu projeto atual (o cliente-servidor) para uma versão puramente frontend, pronta para o GitHub e Vercel:

Fase 1: Preparação e Limpeza do Projeto (na sua nova cópia PAMDE\_Portal\_FrontendOnly)

Crie a Nova Pasta para os Dados no Frontend antes de Deletar a Pasta backend/:  
  
Dentro da sua pasta frontend/, crie uma nova subpasta. O nome que sugeri foi data\_source, mas pode ser dados\_equipamentos, json\_data, ou o que fizer mais sentido para você. Vamos usar frontend/data\_source/ como exemplo.

Copie os Arquivos JSON de Dados:

Vá para a pasta PAMDE\_Portal\_FrontendOnly/backend/data/.

Copie todos os arquivos .json que estão lá (equipamentosvegetais.json, equipamentossafra.json, equipamentosautoclaves.json, equipamentospreparacao.json, equipamentostorres.json, e equipamentosteste.json).

Cole esses arquivos na nova pasta que você criou no passo anterior: PAMDE\_Portal\_FrontendOnly/frontend/data\_source/.

Sobre o requirements.txt:

Você está certo, a imagem mostra que o requirements.txt está dentro da pasta backend/.

Como ele lista as dependências Python do backend (Flask, bcrypt, Flask-CORS, etc.), e nós vamos remover o backend Python, você não precisa salvar este arquivo requirements.txt em outro lugar para o projeto "Frontend Only". Ele se tornará irrelevante para essa versão.

Outros Arquivos na Pasta backend/ a Considerar (e provavelmente descartar para a versão Frontend Only):

\_\_pycache\_\_/: Pasta de cache do Python. Pode ser deletada, não é parte do seu código fonte.

instance/: Contém o pamde\_app.db (banco de dados SQLite). Como não usaremos o backend e o SQLite, essa pasta e seu conteúdo não serão necessários para a versão frontend only.

venv/: Seu ambiente virtual Python. Também não será necessário para a versão puramente frontend.

app.py, data\_loader.py, database.py: Estes são os arquivos principais do backend Python. Não serão usados na versão frontend only.

Deletar a Pasta backend/:

Depois de ter copiado os arquivos JSON de dados para frontend/data\_source/ (ou o nome que escolheu), você pode deletar com segurança toda a pasta PAMDE\_Portal\_FrontendOnly/backend/.

Resumo do que fazer AGORA para a Fase 1:

Na sua cópia do projeto (PAMDE\_Portal\_FrontendOnly):

Crie a pasta frontend/data\_source/.

Copie os arquivos de backend/data/\*.json para frontend/data\_source/.

Delete a pasta backend/ inteira.

Após isso, a estrutura do seu projeto PAMDE\_Portal\_FrontendOnly será focada apenas no frontend, e os dados JSON estarão no lugar certo para serem acessados pelo JavaScript do cliente.

Fase 2: Adaptação do Código Frontend

Autenticação (frontend/js/login.js):

Remover Chamada à API de Login: Delete a lógica fetch que enviava credenciais para /api/login.

Reimplementar Autenticação Local (Simulada):

Volte a ter um array de usuários válidos diretamente no login.js (como tínhamos no início do projeto frontend).

const validUsers = [

{ usuario: "Admin", cadastro: "0000", senha: "000000" },

{ usuario: "Juliano Paiva Teixeira", cadastro: "5941", senha: "521489" },

{ usuario: "Johnnatan Krause Ribeiro Moreno", cadastro: "1218", senha: "964723" }

// Adicione mais se necessário para teste

];

Use code with caution.

JavaScript

A lógica do submit do formulário de login voltará a comparar os inputs com este array.

OU Simplificar Totalmente: Se a autenticação não for um requisito crítico para esta versão "offline", você pode remover a tela de login e fazer com que o index.html redirecione diretamente para o dashboard.html, ou o dashboard.html pode ser sua página inicial. (Manter a simulação de login é bom para manter a estrutura).

Carregamento de Dados dos Equipamentos (frontend/js/checklist.js):

Identificar o Setor: A lógica para pegar setorSelecionado do sessionStorage (definido pelo dashboard.js) permanece a mesma.

Modificar fetchEquipamentosDoSetor(setor):

Esta função não fará mais fetch para uma API do backend.

Ela precisará determinar qual arquivo JSON carregar com base no setor. Crie um mapeamento no checklist.js:

const setorParaArquivoJson = {

"VEGETAIS": "data\_source/equipamentosvegetais.json",

"SAFRA": "data\_source/equipamentossafra.json",

"AUTOCLAVES": "data\_source/equipamentosautoclaves.json",

"PREPARACAO": "data\_source/equipamentospreparacao.json",

"TORRES": "data\_source/equipamentostorres.json",

"TESTE": "data\_source/equipamentosteste.json"

};

Use code with caution.

JavaScript

Use fetch(caminhoDoArquivoJson) para carregar o arquivo JSON local. Isso funciona bem no Live Server e no Vercel.

async function fetchEquipamentosDoSetor(setor) {

const arquivoJson = setorParaArquivoJson[setor.toUpperCase()];

if (!arquivoJson) {

// Tratar erro: setor desconhecido

return;

}

try {

const response = await fetch(arquivoJson);

if (!response.ok) throw new Error(`Não foi possível carregar ${arquivoJson}`);

const dataJsonCompleto = await response.json();

// AGORA VEM A PARTE DE NAVEGAR NA ESTRUTURA DO JSON

// E MONTAR 'listaDeEquipamentos' no mesmo formato que a API retornava

listaDeEquipamentos = parseJsonParaListaDeEquipamentos(dataJsonCompleto, setor.toUpperCase());

// ... resto da lógica (loadingMessage, exibirEquipamentoAtual, etc.)

} catch (error) { /\* ... tratar erro ... \*/ }

}

Use code with caution.

JavaScript

Nova Função parseJsonParaListaDeEquipamentos(dataJsonCompleto, setorPrincipal):

Esta será a função mais complexa de adaptar. Você precisará recriar a lógica de navegação dos JSONs que tínhamos no data\_loader.py, mas agora em JavaScript.

Ela receberá o objeto JSON completo e o nome do setor.

Iterará pelos níveis do JSON (subsetor1, subsetor2, componente) e, para cada motor, criará um objeto com as mesmas chaves que a API do backend retornava (e que a tabela equipamentos usava: tag\_motor, setor\_principal, subsetor\_nivel1, etc., e todos os campos nominais).

Retornará o array listaDeEquipamentos.

Dica: Você pode se basear nas funções process\_standard\_structure\_json e process\_safra\_json do data\_loader.py para a lógica de iteração. A sintaxe de loops e acesso a propriedades de objetos é similar entre Python e JavaScript.

Coleta e Armazenamento de Dados da Inspeção (frontend/js/checklist.js):

A lógica atual que armazena os dados em dadosColetadosInspecao (incluindo o objeto File da foto e o previewDataUrl) pode ser mantida.

Lógica de "Não se Aplica" / "Não Medido" (frontend/js/checklist.js e checklist.html):

Toda a lógica que implementamos para os checkboxes N/A e N/M pode ser mantida, pois é puramente frontend.

Geração de Relatório PDF (frontend/js/checklist.js na função handleFinalizarInspecao):

Escolher uma Biblioteca JS para PDF:

jsPDF (https://github.com/parallax/jsPDF) é popular. Para tabelas, jsPDF-AutoTable (https://github.com/simonbengtsson/jsPDF-AutoTable) é um plugin essencial.

pdfmake (http://pdfmake.org/) é outra opção poderosa baseada em declaração de conteúdo.

Incluir a Biblioteca: Adicione a(s) biblioteca(s) escolhida(s) ao seu projeto (geralmente via tag <script> no HTML ou importando se você for usar um bundler no futuro, mas para começar, a tag script é mais simples).

Implementar a Geração:

Na handleFinalizarInspecao, após obter inspecoesValidas:

Crie uma nova instância da biblioteca de PDF (ex: const doc = new jsPDF();).

Adicione o título, informações do inspetor (pegar do localStorage), data/hora.

Itere sobre inspecoesValidas. Para cada equipamento inspecionado:

Formate os dados nominais e medidos para uma tabela.

Adicione o "Parecer" (que ainda precisaremos implementar a lógica de cálculo).

Fotos: Este é o ponto mais tricky. Com jsPDF, você pode adicionar imagens a partir de DataURLs. O foto\_equipamento\_previewDataUrl que você já salva é um DataURL!

Adicione observações.

Use os métodos da biblioteca para adicionar texto, tabelas, imagens ao PDF.

Finalmente, use doc.save('relatorio\_inspecao.pdf'); para acionar o download do PDF no navegador do usuário.

Lógica de Não Conformidades / "Parecer" (frontend/js/checklist.js):

Implementar a função calcularNaoConformidades(dadosNominais, dadosMedidos) como planejado.

Chamar essa função em coletarDadosDoFormulario para que o parecer seja salvo junto com os outros dados da inspeção e possa ser usado no relatório PDF.

Fase 3: Testes e Implantação

Testar Localmente: Use o Live Server extensivamente para testar todas as funcionalidades.

Preparar para o GitHub:

Crie um arquivo .gitignore na raiz do PAMDE\_Portal\_FrontendOnly para ignorar arquivos desnecessários (ex: se o VS Code criar uma pasta .vscode). Um .gitignore simples pode ser:

.vscode/

node\_modules/

\*.env

Use code with caution.

Enviar para o GitHub: Crie um repositório no GitHub e envie seu projeto.

Implantar no Vercel:

Crie uma conta no Vercel (se ainda não tiver).

Importe seu projeto do GitHub.

Configure o Vercel:

Framework Preset: Selecione "Other" ou deixe o Vercel tentar detectar.

Build Command: Pode deixar em branco (ou npm install && npm run build se você adicionar um package.json com scripts, mas para HTML/CSS/JS puros, não é estritamente necessário inicialmente).

Output Directory: Se a raiz do seu site é a pasta frontend/, você pode precisar ajustar isso ou configurar o "Root Directory" para frontend. O Vercel geralmente é bom em detectar o index.html.

Environment Variables: Não precisaremos de variáveis de ambiente complexas se não houver backend.

Faça o deploy. O Vercel fornecerá uma URL pública para seu projeto.

Este plano é abrangente, mas dividindo em fases e passos menores, se torna mais gerenciável. A parte mais trabalhosa será adaptar fetchEquipamentosDoSetor para parsear os JSONs e implementar a geração de PDF no cliente.