📘 Estudo de Caso — *Pocket Weather Dashboard*

🎯 Objetivo

Desenvolver um **mini-dashboard de clima** que permite ao usuário consultar o clima atual de uma cidade, exibindo:

- Nome da cidade
- Temperatura
- Condição do tempo (ex.: nublado, ensolarado)
- Ícone representativo do clima

> O alvo aqui não é a qualidade visual avançada, mas a **estruturação correta do código**, clareza e uso adequado das ferramentas.

🧠 Conceitos Exercitados

- Estrutura semântica HTML
- Estilização com CSS simples + responsividade básica
- Manipulação de DOM com JavaScript
- Requisição a API pública com `fetch()` *(simples e realista)*
- Organização do projeto conforme o setup criado
- Uso de variáveis de ambiente para armazenar a API_KEY
- Console logs para depuração
- Separação de responsabilidades (HTML/CSS/JS)

✨ Requisitos Funcionais

1. Campo de texto para digitar a cidade
2. Botão “Buscar clima”
3. Exibição das informações retornadas
4. Loader simples (mensagem “Carregando...”) enquanto busca
5. Mensagem de erro quando a cidade não existir

🗂 Estrutura mínima esperada

```
src/
├── index.html
├── style.css
└── script.js
.env
```

🌍 API Sugerida (gratuita)

OpenWeatherMap Free API

Endpoint:

```
https://api.openweathermap.org/data/2.5/weather?q=CITY&appid=APIKEY&units=metric&lang=pt_br
```

Variável no `.env`:

```
WEATHER_API_KEY=SUA_CHAVE_AQUI
```

> O servidor Node já lê `.env`, então os alunos devem usar a estratégia simples: armazenar a chave no backend e injetar em script JS com variável global (ex.: `window.API_KEY = process.env.WEATHER_API_KEY`) — mas como o foco é apenas frontend e aprendizado, você pode simplificar deixando-os copiar a key diretamente no JS apenas para o exercício.
> Depois, há espaço para discutir boas práticas de segurança.

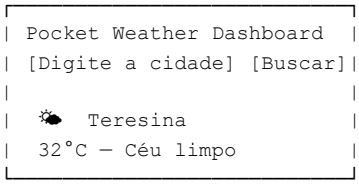
✅ Critérios de Avaliação

Critério	Avaliação
Organização do projeto	Estrutura de arquivos e nomeação
HTML semântico	Uso adequado de tags e atributos
CSS organizado	Classes claras, layout legível
Responsividade mínima	Layout adaptável (flex/grid simples)
JavaScript claro	DOM, funções, tratamento de erro
API funcionando	Retorno e apresentação dos dados
Boas práticas	ESLint sem erros, Prettier aplicado
Entrega no GitHub	Repositório organizado

💡 Requisitos Extras (opcionais para desafiar)

- Tema claro/escuro
- Salvar última cidade pesquisada no `localStorage`
- Transição/animação suave quando aparecer o resultado
- Ícones animados para clima (por exemplo, https://erikflowers.github.io/weather-icons/)

🎨 Sugestão mínima de layout



Seja minimalista, clean e funcional.

🔍 Reflexão técnica para os alunos

Após concluir, os estudantes devem responder:

1. Qual foi a maior dificuldade técnica?
2. Como o ESLint/Prettier ajudou?
3. O que fariam diferente se fosse um projeto maior?
4. Qual padrão de organização de código usaram?
5. Como fariam cache dos dados da API?

Isso incentiva pensamento crítico e maturidade técnica.

🧭 Pontos a serem considerados

Cada equipe deve apresentar:

- O código está bem escrito e organizado?
- O fluxo da aplicação é coerente?
- Debugs no console
- Estrutura do repositório
- Execução local/remota

> Incentive qualidade sobre quantidade — foco em **clareza, organização e padrão profissional**.