Com base no **escopo do projeto** e no que vocês já estão fazendo (sistema de mapeamento de pontos de coleta de óleo), aqui vai **um passo a passo simples** mostrando **como atender a todos os requisitos da forma mais fácil e prática possível**, sem complicar o código:

---

## □ 1. Interface com o usuário (Front-end)

**Objetivo:** ter uma página que o usuário possa interagir — ver o mapa, ler informações, etc.
**Forma mais simples de fazer:**

- Criar um **site em HTML + CSS + JavaScript** básico.
- Usar o **Leaflet.js** (biblioteca gratuita) para mostrar o mapa.
- Criar três seções:
  - **Página inicial:** explicando o problema do óleo e os objetivos do site.
  - **Mapa interativo:** exibe os pontos de coleta.
  - **Página educativa:** dicas de reutilização e descarte correto.

💡 **Dica:** o Leaflet já traz exemplos prontos — é só copiar o código base e inserir suas coordenadas de coleta.

---

## 🌐 2. Integração com API pública ou privada

**Objetivo:** mostrar dados que venham de uma API (ou simular isso).
**Forma mais fácil:**

- Usar a **API de mapas (Leaflet + OpenStreetMap)** — já conta como uma API pública.
- Opcionalmente, consumir dados de outra API de meio ambiente (ou um arquivo CSV hospedado no GitHub simulando uma API).

💡 **Exemplo:** criar um arquivo `pontos.json` com as localizações e carregá-lo no site usando `fetch('pontos.json')`.

---

## 🧠 3. Contexto de Componentes Distribuídos

**Objetivo:** mostrar que o sistema pode se comunicar entre partes (front e back-end).
**Forma mais simples:**

- Criar um **pequeno back-end em Python (Flask ou FastAPI)** com uma rota `/pontos` que devolve os dados em JSON.
- O front-end (HTML + JS) faz uma requisição `fetch("http://localhost:5000/pontos")` e mostra os marcadores no mapa.

💡 Assim, vocês já demonstram "componentes distribuídos": um servidor que envia dados e um cliente que consome.

---

## ❁ 4. Conceitos fundamentais (Design Patterns, Microserviços, Kubernetes, etc.)

**Objetivo:** mostrar que entenderam boas práticas de desenvolvimento.
**Forma mais simples:**

- **Design Pattern:** aplicar o padrão **MVC (Model-View-Controller)**, separando:
  - Model → arquivo com os dados ou funções do back-end.
  - View → páginas HTML.
  - Controller → rota Flask que conecta tudo.
- **Microserviço:** manter o serviço de pontos de coleta separado (por exemplo, `pontos_service.py`).
- **Kubernetes:** apenas **mencionar** que, se o projeto crescesse, poderia ser colocado em contêineres (Docker) e orquestrado no Kubernetes.

💡 Vocês **não precisam implementar o Kubernetes de fato**, só mostrar no relatório que o sistema é modular e *poderia* ser distribuído.

---

## ☑ Resumo do jeito mais fácil

| Requisito | Solução prática |
|---|---|
| Interface (front-end) | HTML + CSS + JS + Leaflet (mapa interativo) |
| Integração com API | Leaflet/OpenStreetMap ou arquivo JSON simulado |
| Componentes distribuídos | Flask enviando dados e front-end consumindo |
| Conceitos fundamentais | MVC + menção a microserviço e Kubernetes |

---