Qualidade de Software

Objetivo: Criar uma barreira de proteção no GitHub que impeça QUALQUER erro de estrutura ou de arquivos no site.

Parte 1: A Estrutura

Em vez de um arquivo só, vamos simular um projeto real com pastas. Organizar o projeto no VS Code exatamente assim:

1. Criar uma pasta chamada `src/`.
2. Mover o `index.html` para dentro de `src/`.
3. Criar uma pasta chamada `css/` dentro de `src/`.
4. Mover o `style.css` para dentro de `src/css/`.
5. **Ajuste de Código:** É necessário atualizar o link do CSS no HTML para: `<link rel="stylesheet" href="css/style.css">`.

Parte 2: A Esteira de Verificação Tripla

Agora o arquivo YAML precisa ser atualizado para refletir essa nova estrutura. Você deve configurar o Workflow p.

O que o Workflow deve fazer:

1. **Checkout** (Action oficial).
2. **Verificar Pasta:** Um comando `run` que verifica se a pasta `src` existe.
3. **Verificar CSS:** Um comando `run` que verifica se o arquivo está no caminho novo (`test -f src/css/style.css`).
4. **Linting Profissional:** Usar a Action `htmlhint` apontando para o caminho novo: `target_pattern: 'src/index.html'`.

Parte 3: O Teste

Lista de **Erros Propositalmente Cometidos**. É necessário realizar um por um, dar `push`, ver o erro, tirar print do erro no log e depois consertar.

Lista de Sabotagem:

1. **Erro de Tag:** Apague o fechamento da tag `<head>`.
2. **Erro de Atributo:** Coloque um atributo que não existe (ex: `<div xxxx="bbb>`). O HTMLHint vai reclamar?

3. **Erro de Caminho:** Renomeie a pasta `css` para `styles.`
4. **Erro de Texto:** O HTML deve obrigatoriamente ter uma tag `<footer>` com o nome do aluno. Se não tiver, não será aceito.

Parte 4: O Relatório de Auditoria

Desafio Final: Adicionar um passo final no YAML que use o comando `echo` para imprimir uma mensagem personalizada no log se tudo passar:

"O código do aluno [NOME] foi auditado, está seguindo as normas corretas e está pronto para o deploy!"