**Relatório de Alterações — Função carregarPagina**

1. **Verificação da existência do container antes de injetar conteúdo HTML**

**Descrição:**

Antes de inserir o HTML carregado no elemento principal da aplicação (#conteudo-principal), foi implementada uma verificação para garantir que o elemento existe no DOM.

**Trecho relevante:**

const container = document.getElementById("conteudo-principal");

if (container) {

container.innerHTML = html;

} else {

console.warn("Elemento #conteudo-principal não encontrado.");

}

**Justificativa:**  
Evita erros JavaScript que paralisam a execução caso o container não exista, aumentando a resiliência da aplicação e facilitando o diagnóstico em ambiente de desenvolvimento.

**2. Remoção dos scripts inline do HTML injetado e carregamento controlado dos scripts externos**

**Descrição:**  
Para evitar rejeição do conteúdo por conter scripts inline — prática que reduz vulnerabilidades de segurança como XSS — o conteúdo HTML carregado não deve conter tags <script> embutidas.

Dessa forma, toda a lógica JavaScript necessária é externalizada para arquivos .js específicos carregados sob demanda via a função carregarScriptsSequencial, mapeada pelo objeto pageToScriptMap.

**Trecho relevante:**

.then((html) => {

// Verificação defensiva contra scripts inline

const parser = new DOMParser();

const doc = parser.parseFromString(html, "text/html");

const inlineScripts = doc.querySelectorAll("script:not([src])");

if (inlineScripts.length > 0) {

console.error(

"HTML rejeitado: contém scripts inline não permitidos."

);

document.getElementById("conteudo-principal").innerHTML = `

<div class="alert alert-danger">

<i class="fas fa-exclamation-triangle"></i>

Erro: Conteúdo rejeitado por conter scripts inline.

</div>`;

return;

}

**Impacto na arquitetura:**

* Eliminação de scripts inline no HTML permite a aplicação de políticas CSP mais rigorosas.
* Melhora a separação de responsabilidades entre estrutura (HTML) e comportamento (JS).
* Facilita manutenção e reutilização dos scripts.

**3. Transferência da inicialização para os scripts externos**

**Descrição:**  
Funções de inicialização, antes escritas inline em cada página HTML (exemplo: document.addEventListener("DOMContentLoaded", ...) ou chamadas diretas a funções init()), foram removidas dos arquivos HTML e incorporadas nos scripts JavaScript externos correspondentes.

**Justificativa:**

* Consistência no carregamento assíncrono e modular da aplicação.
* Evita execução prematura de código antes da injeção do conteúdo HTML.
* Garante que a lógica de inicialização ocorre somente após a carga e inserção do conteúdo.

**4. Log informativo controlado de sucesso no carregamento da página**

**Descrição:**  
Foi adicionado um log com console.info logo após a injeção bem-sucedida do conteúdo HTML, indicando que a página foi carregada.

**Trecho exemplo:**

console.info(`[FiberGuardian] Página '${pagina}' carregada com sucesso.`);

**Benefício:**  
Facilita o debug e monitoramento do fluxo de navegação e carregamento durante o desenvolvimento.

**5. Tratamento e log aprimorados para carregamento de scripts associados**

**Descrição:**

A função verifica a existência de scripts associados no mapa pageToScriptMap antes de tentar carregá-los sequencialmente, emitindo warnings quando scripts não são encontrados para determinada página.

**Justificativa:**  
Previne erros inesperados e melhora a manutenção, alinhando-se ao princípio KISS (Keep It Simple, Stupid).

**6. Tratamento de erros com fallback visual e log**

**Descrição:**  
Ao ocorrer falha no fetch ou na injeção do conteúdo HTML, além do log de erro no console, uma mensagem visual de alerta é exibida na área principal para informar o usuário final.

**Trecho exemplo:**

container.innerHTML = `<div class="alert alert-danger">

<i class="fas fa-exclamation-triangle"></i>

Erro ao carregar conteúdo: ${erro.message}

</div>`;

**Benefício:**

Melhora a usabilidade e experiência, fornecendo feedback claro e seguro em conformidade com as recomendações OWASP.

**Considerações finais**

Essas mudanças tornam o carregamento dinâmico de páginas mais robusto, seguro e alinhado às boas práticas de desenvolvimento frontend moderno, incluindo:

* Modularidade e separação clara de responsabilidades (HTML vs JS).
* Aumento da segurança via remoção de scripts inline e suporte a CSP.
* Melhora na experiência de usuário final e facilidade de diagnóstico para desenvolvedores.