**🧱 Arquitetura da Tela de Login (MPA + JS Modular)**

**🔹 1. Estrutura Frontend**

Você optou por uma arquitetura **MPA (Multi Page Application)** com páginas HTML estáticas servidas diretamente, e interatividade implementada com **JavaScript Vanilla modular** via IIFE. Isso traz benefícios claros:

| **Critério** | **Abordagem** | **Justificativa** |
| --- | --- | --- |
| **Separação de responsabilidades (SRP)** | fiberguardian\_utils.js contém utilitários genéricos, tela\_login.js é específico da tela | Evita acoplamento e favorece reuso e testes |
| **Encapsulamento (KISS/DRY)** | Uso de IIFE para namespace FiberGuardian | Impede vazamento de variáveis no escopo global |
| **Responsividade e acessibilidade** | Uso de **Bootstrap 5** + validação nativa do HTML (required, type="email") | Interface responsiva e com fallback nativo do browser |
| **UX defensiva** | Validação manual com isEmailValido() antes de submeter | Minimiza chamadas desnecessárias ao backend |

**🔹 2. JavaScript Modular Seguro**

Você adotou corretamente um padrão de **namespace único global controlado (window.FiberGuardian)**, garantindo:

* **Modularização com isolamento lógico**: funções utilitárias (CSRF, cookies, validação) separadas da lógica da tela.
* **Injeção de dependência explícita**: FiberGuardian.TelaLogin depende de FiberGuardian.Utils, mas essa dependência é clara e controlada.

O uso de credentials: "include" no fetch é **essencial** para manter a sessão autenticada via cookies HttpOnly.

**🔐 Segurança na Arquitetura**

Você aplicou diversas medidas de segurança alinhadas com as recomendações da OWASP Web Security Top 10:

**🔸 CSRF (Cross-Site Request Forgery)**

* Implementado via:

js

CopiarEditar

const csrfToken = await FiberGuardian.Utils.obterTokenCsrf();

headers: { "X-XSRF-TOKEN": csrfToken }

* Boa prática de defesa **double submit cookie** se o servidor estiver configurado para:
  + Enviar o token como cookie (Set-Cookie: XSRF-TOKEN=...)
  + Exigir o header "X-XSRF-TOKEN" correspondente

✅ **Comentário positivo:** sua função obterTokenCsrf lida com erros de forma robusta, mantendo rastreabilidade em logs com console.error.

**🔸 Cookies de Sessão HttpOnly**

* Sua arquitetura espera que a sessão seja mantida via cookies, com:

js

CopiarEditar

credentials: "include"

* Isso pressupõe que o Spring Boot está configurado com:

java

CopiarEditar

http.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.IF\_REQUIRED);

http

.csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())

* Esses cookies não são acessíveis via JS (HttpOnly) — proteção contra **XSS**.

**🔸 Validação e Normalização de Entrada**

* Você implementou normalizarEmail() e validou com regex antes de submeter ao backend.

js

CopiarEditar

email = email.trim().toLowerCase();

/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

* Isso evita **input inconsistente** e reduz **ambiguidade de identidade**.

**🔸 Anti-XSS e Regras de UX**

* Uso de innerText ou JSON seguro (sem eval, innerHTML, etc.)
* Formulário não tenta processar HTML inserido pelo usuário
* Redirecionamento pós-login controlado via:

js

CopiarEditar

window.location.href = "index.html";

**📐 Considerações Arquiteturais**

| **Ponto Forte** | **Justificativa** |
| --- | --- |
| **MPA + Vanilla JS** | Mais seguro por padrão, sem dependência de frameworks ou bundlers |
| **IIFE + namespace único** | Padrão ideal para código legível e isolado |
| **Fetch + CSRF + Session** | Fluxo moderno compatível com Spring Security e CORS |
| **Fallback nativo de segurança do browser** | Campos required, type="email", autofocus aproveitam capacidades nativas do HTML5 |

**📌 Sugestões de melhoria futura**

1. **UX progressiva com feedback no botão** (ex: loading spinner):

js

CopiarEditar

btn.disabled = true;

btn.innerHTML = "Entrando...";

1. **Throttle no envio de formulários** para evitar brute-force (em JS e no backend).
2. **Headers de segurança no backend**: Content-Security-Policy, X-Frame-Options, etc.
3. **Auditoria do backend com logs claros para falhas de login** (timestamp, IP, user-agent).

**A arquitetura foi desenhada para ser segura e modular desde o início, com mecanismos de proteção contra CSRF e XSS, e baseada em cookies de sessão. Funcionalidades de rastreamento e auditoria estão previstas para fases posteriores, seguindo uma abordagem incremental e de hardening progressivo após validação da autenticação básica.**