Exatamente — sua análise está perfeita. A **melhor estratégia** é:

* ✅ Implementar a lógica de verificação periódica em **FiberGuardian.Utils** como uma **função privada** (não exportada),
* ✅ Iniciar o monitoramento **em páginas protegidas**, chamando uma função initSessaoWatcher() no DOMContentLoaded,
* 🚫 **Não expor como função pública** se o objetivo for encapsular o comportamento automático.

**📐 Por que colocar isso em FiberGuardian.Utils?**

| **Motivo** | **Justificativa** |
| --- | --- |
| 🔁 Reutilização | Todas as páginas protegidas precisam desse comportamento |
| 🔒 Encapsulamento | A lógica de verificação não precisa estar acessível externamente |
| ✅ SRP | Utilitários cuidam da comunicação e sessão — responsabilidade clara |

**✅ Como estruturar**

**🔹 Dentro de fiberguardian\_utils.js**

js

CopiarEditar

(function () {

window.FiberGuardian = window.FiberGuardian || {};

FiberGuardian.Utils = (function () {

const API\_BASE\_URL = "/fiberguardian";

function getApiUrl(path) {

return `${API\_BASE\_URL}${path}`;

}

function getCookie(nome) {

const cookies = document.cookie.split("; ");

for (const cookie of cookies) {

const [chave, valor] = cookie.split("=");

if (chave === nome) {

return decodeURIComponent(valor);

}

}

return null;

}

function isEmailValido(email) {

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

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

}

async function obterNovoToken() {

const resposta = await fetch(getApiUrl("/csrf-token"), {

method: "GET",

credentials: "include",

});

const dados = await resposta.json();

if (!dados.token) {

throw new Error("Token CSRF não retornado pelo servidor.");

}

return dados.token;

}

async function obterTokenCsrf() {

const tokenExistente = getCookie("XSRF-TOKEN");

if (tokenExistente) {

return tokenExistente;

}

return await obterNovoToken();

}

// 🔒 Função privada: verifica se a sessão ainda está ativa

async function verificarSessao() {

try {

const resposta = await fetch(getApiUrl("/sessao/valida"), {

method: "GET",

credentials: "include",

});

if (resposta.status === 401 || resposta.status === 403) {

alert("Sua sessão expirou. Você será redirecionado para o login.");

window.location.href = "tela\_login.html";

}

} catch (erro) {

console.warn("Erro ao verificar sessão:", erro);

}

}

// 🔒 Função privada: inicia verificação periódica

function iniciarMonitoramentoSessao() {

setInterval(verificarSessao, 5 \* 60 \* 1000); // a cada 5 minutos

}

// ✅ Função pública para inicializar o watcher

function iniciarWatcherDeSessao() {

iniciarMonitoramentoSessao();

}

return {

getCookie,

isEmailValido,

obterTokenCsrf,

obterNovoToken,

getApiUrl,

iniciarWatcherDeSessao, // única função pública nova

};

})();

})();

**✅ Como ativar o monitoramento**

Em **cada página protegida**, no DOMContentLoaded:

html

CopiarEditar

<script>

document.addEventListener("DOMContentLoaded", function () {

FiberGuardian.Utils.iniciarWatcherDeSessao();

FiberGuardian.TelaPrincipal.init(); // ou outro módulo da página

});

</script>

**🧠 Vantagem adicional**

Essa abordagem permite:

* ✅ Reutilização segura e encapsulada
* ✅ Fácil desativação (ex: páginas públicas não chamam o watcher)
* ✅ Redução de erros silenciosos pós-timeout

Se quiser, posso revisar seu backend para sugerir o endpoint /sessao/valida leve e seguro. Deseja isso?

**Você disse:**

ahhh eu preciso implementar o endpoint no backend tambem! "/sessao/valida"

**O ChatGPT disse:**

Exatamente! Para que o fetch('/sessao/valida') funcione corretamente, você precisa criar um endpoint leve e seguro no backend, que apenas **verifica se a sessão está ativa e autenticada**.

**✅ Objetivo do endpoint /sessao/valida**

* ✅ Deve **exigir autenticação** (proteção por sessão),
* ✅ Retornar 200 OK se a sessão estiver válida (usuário autenticado),
* ✅ Retornar 401 Unauthorized se a sessão estiver ausente ou expirada,
* ✅ Não precisa retornar corpo nem acessar banco.

**📄 Exemplo da implementação no Spring Boot**

java

CopiarEditar

@RestController

@RequestMapping("/sessao")

public class SessaoController {

@GetMapping("/valida")

public ResponseEntity<Void> validarSessao(HttpServletRequest request) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

boolean autenticado = auth != null &&

auth.isAuthenticated() &&

!(auth instanceof AnonymousAuthenticationToken);

return autenticado ? ResponseEntity.ok().build()

: ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

}

}

**🔐 Segurança do endpoint**

Esse endpoint **deve estar protegido** por Spring Security para que:

* Se a sessão tiver expirado, o Spring Security **automaticamente devolve 401 ou 403**.
* Se a sessão estiver ativa, ele retorna 200.

Ou seja, **não precisa de lógica extra no controller para verificar o timeout** — Spring cuida disso.

**🛡️ Segurança adicional**

* Esse endpoint **não expõe dados do usuário** (bom para segurança).
* Pode ser usado com fetch(..., { credentials: "include" }) no frontend.
* Não precisa de CSRF, pois é GET e **não altera estado**.