**✅ TL;DR**

**Não, você não precisa solicitar um novo token CSRF a cada troca de página**, **desde que** o servidor configure o token CSRF como um **cookie acessível via JavaScript (não HttpOnly)** e ele seja mantido entre páginas. Seu JavaScript pode reutilizá-lo tranquilamente.

**🔐 Entendendo o funcionamento com base no Spring Security**

Se você está usando Spring Security com a configuração típica de:

java

CopiarEditar

.csrf()

.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())

Então o backend:

* Gera um token CSRF novo **por sessão ou por request** (dependendo da config),
* **Envia esse token no header Set-Cookie**, por exemplo:

http

CopiarEditar

Set-Cookie: XSRF-TOKEN=abc123; Path=/; Secure; SameSite=Lax

Esse token:

* É **armazenado no cookie do navegador**, visível ao JavaScript (porque *HttpOnly = false*),
* Pode ser lido com sua função getCookie("XSRF-TOKEN"),
* E enviado no header "X-XSRF-TOKEN" pelo seu fetch().

**🧠 Situação no seu MPA**

**📦 Quando você troca de página em uma MPA:**

* A página é **completamente recarregada** (não é um SPA),
* Todos os arquivos JS são **descarregados e carregados novamente**,
* As **variáveis em memória do JS anterior são perdidas**,
* **Cookies persistem normalmente**, inclusive o XSRF-TOKEN.

**🛠 Como reutilizar o token sem nova chamada**

Basta alterar sua função obterTokenCsrf() para **tentar primeiro ler do cookie**, antes de fazer fetch():

js

CopiarEditar

async function obterTokenCsrf() {

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

if (tokenExistente) {

return tokenExistente;

}

// fallback defensivo, só chama se não encontrar no cookie

try {

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

method: "GET",

credentials: "include",

});

if (!resposta.ok) {

throw new Error(`Erro ao obter token CSRF: ${resposta.statusText}`);

}

const dados = await resposta.json();

if (!dados.token) {

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

}

return dados.token;

} catch (erro) {

console.error("Falha ao obter token CSRF:", erro);

throw erro;

}

}

Assim você:

* **Evita chamadas redundantes** ao backend a cada nova página,
* **Aproveita o token já disponível** via cookie (persistente entre páginas),
* E mantém uma fallback defensiva caso o cookie tenha expirado ou não exista.

**⚠️ Considerações de Segurança**

* O cookie XSRF-TOKEN **não deve ser HttpOnly**, justamente para que o JS possa ler.
* O valor do cookie **não precisa ser secreto** (o segredo é a verificação do header + cookie no servidor).
* Você **deve continuar enviando o header "X-XSRF-TOKEN"** em chamadas fetch, mesmo em páginas subsequentes.

**📌 Conclusão**

**✅ Você pode (e deve) reutilizar o token CSRF armazenado em cookie, e só fazer nova requisição se o cookie não existir.**

Essa prática:

* Melhora performance (menos chamadas),
* Evita overfetching,
* E continua segura, desde que seu backend verifique **cookie + header**.