Perfeito. Com base na análise do seu código e no erro relatado (Invalid CSRF token found for https://localhost:8443/fiberguardian/login), o diagnóstico técnico é o seguinte:

**⚠️ Problema Central**

O token CSRF está correto no cabeçalho, **mas o cookie JSESSIONID da sessão onde o token foi gerado não está sendo enviado na requisição POST para o endpoint de login**. Isso causa a criação de uma **nova sessão**, que tem um **token CSRF diferente**, invalidando a requisição.

**🔍 Diagnóstico Detalhado**

**1. Você usa credentials: "include" corretamente**

Tanto na requisição GET /csrf-token quanto na POST /login, você especifica:

js

CopiarEditar

credentials: "include"

✅ Isso instrui o fetch() a enviar e receber cookies mesmo em requisições cross-origin com HTTPS — ótimo.

**2. O token CSRF vem da sessão correta**

Seu backend gera o token via CookieCsrfTokenRepository, o qual associa o token à **sessão atual** (JSESSIONID). Ao fazer o GET /csrf-token, o backend:

* Cria uma sessão JSESSIONID
* Gera e envia o token CSRF via cookie XSRF-TOKEN
* Retorna o token no corpo da resposta também (como JSON)

Até aqui tudo certo.

**3. O problema: cookie JSESSIONID não é enviado na requisição de login**

Esse é o motivo da mensagem de debug no Spring Security:

txt

CopiarEditar

DEBUG o.s.security.web.csrf.CsrfFilter: Invalid CSRF token found

Verificações que confirmam isso:

* No AuthController, o log imprime request.getSession(false).getId() → provavelmente null ou uma nova sessão.
* Consequentemente, o token CSRF da requisição POST é **válido**, mas **não pertence à sessão da requisição atual**.

**✅ Soluções Recomendadas**

**✅ 1. Certifique-se que o frontend está servindo via HTTPS**

Como seus cookies têm Secure = true, o navegador **só envia os cookies JSESSIONID e XSRF-TOKEN se a página HTML tiver sido carregada via https://... também.**

**Verifique:**

* Sua página HTML de login está servida por https://localhost:8080?
* O certificado usado no frontend é confiável pelo navegador?
* O SameSite=Lax não impede envio em POST de form.submit() com fetch(), mas **o navegador ainda exige HTTPS para cookies Secure**.

**✅ 2. Verifique no navegador se o cookie JSESSIONID foi setado e está sendo enviado**

Abra o DevTools > aba **Network**:

* Requisição GET /csrf-token
  + Cookie JSESSIONID é criado?
* Requisição POST /login
  + O **mesmo** JSESSIONID aparece no header Cookie?

Se não estiver, a origem do HTML (localhost:8080) pode estar sendo considerada **outra origem** (por CORS + certificado), mesmo com credentials: include.

**🔧 Ajustes Técnicos Adicionais**

**🔹 1. Verifique se a sessão é criada corretamente em /csrf-token**

Confirme que o endpoint /csrf-token chama request.getSession(true) para **forçar criação da sessão**, ou ao menos algum filtro anterior faz isso. Se a sessão não existir nesse momento, o JSESSIONID pode não ser setado.

**🔹 2. Confirme o domínio dos cookies**

O cookie JSESSIONID deve ser visível para o navegador no domínio e porta da requisição de login:

plaintext

CopiarEditar

Set-Cookie: JSESSIONID=...; Path=/; Secure; SameSite=Lax

Verifique:

* Domínio correto (sem Domain=... fixo errado)
* Porta coerente (localhost:8080 e backend em localhost:8443)

**🧪 Teste rápido para depuração**

**Em DevTools:**

1. Vá em Application > Cookies > <https://localhost:8443>
2. Veja se **JSESSIONID está presente** após GET /csrf-token
3. Veja se ele é enviado em POST /login

Se o cookie está presente mas **não é enviado**, é quase certo que:

* **O HTML foi carregado por HTTP**, não HTTPS
* Ou **as origens são distintas e o navegador bloqueou o envio por política de segurança**

**✅ Conclusão**

📌 A causa raiz **não está no código do Spring ou do frontend JS**, mas sim na **negociação de cookies com HTTPS e CORS**.  
🛠️ Para resolver:

1. **Garanta que o HTML está servido via HTTPS (localhost:8080 com certificado confiável)**
2. **Use sempre credentials: "include" (você já usa)**
3. **Confirme no navegador se o cookie JSESSIONID está sendo enviado no POST**

Se quiser, posso ajudar a criar um pequeno servidor HTTPS local para servir o HTML da forma correta com localhost:8080. Deseja isso?

Perguntar ao ChatGPT

**Você disse:**

só uma observação: o o log que prime request.getSession(false).getId() sequer aparece, aparentemente o controller nem é chamado: veja o log do spring completo: --- 2025-07-01T10:44:31.909-03:00 INFO 4772 --- [fiberguardian] [nio-8443-exec-9] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet' 2025-07-01T10:44:31.909-03:00 INFO 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet' 2025-07-01T10:44:31.909-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Detected StandardServletMultipartResolver 2025-07-01T10:44:31.909-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Detected AcceptHeaderLocaleResolver 2025-07-01T10:44:31.910-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Detected FixedThemeResolver 2025-07-01T10:44:31.911-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Detected org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator@11703cc8 2025-07-01T10:44:31.911-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Detected org.springframework.web.servlet.support.SessionFlashMapManager@1c294a17 2025-07-01T10:44:31.912-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : enableLoggingRequestDetails='false': request parameters and headers will be masked to prevent unsafe logging of potentially sensitive data 2025-07-01T10:44:31.912-03:00 INFO 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms 2025-07-01T10:44:31.930-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.security.web.FilterChainProxy : Securing GET /fiberguardian/csrf-token 2025-07-01T10:44:31.932-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.s.w.a.c.ChannelProcessingFilter : Request: filter invocation [GET /fiberguardian/csrf-token]; ConfigAttributes: [REQUIRES\_SECURE\_CHANNEL] 2025-07-01T10:44:31.957-03:00 TRACE 4772 --- [fiberguardian] [nio-8443-exec-9] s.s.w.c.CsrfTokenRequestAttributeHandler : Wrote a CSRF token to the following request attributes: [\_csrf, org.springframework.security.web.csrf.CsrfToken] 2025-07-01T10:44:31.957-03:00 TRACE 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.security.web.csrf.CsrfFilter : Did not protect against CSRF since request did not match And [CsrfNotRequired [TRACE, HEAD, GET, OPTIONS], Not [Or [Deferred [Mvc [pattern='/fiberguardian/csrf-token'], Ant [pattern='/fiberguardian/csrf-token']]]]] 2025-07-01T10:44:31.960-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.s.w.a.AnonymousAuthenticationFilter : Set SecurityContextHolder to anonymous SecurityContext 2025-07-01T10:44:31.962-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.security.web.FilterChainProxy : Secured GET /fiberguardian/csrf-token 2025-07-01T10:44:31.965-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : GET "/fiberguardian/csrf-token", parameters={} 2025-07-01T10:44:31.966-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to edu.entra21.fiberguardian.controller.CsrfController#getCsrfToken(HttpServletRequest) 2025-07-01T10:44:31.988-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Using 'application/json', given [\*/\*] and supported [application/json, application/\*+json] 2025-07-01T10:44:31.989-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Writing [{headerName=X-XSRF-TOKEN, parameterName=\_csrf, cookieName=XSRF-TOKEN, token=ukA\_\_W31BiJdLO2DRespWQhc (truncated)...] 2025-07-01T10:44:31.998-03:00 DEBUG 4772 --- [fiberguardian] [nio-8443-exec-9] o.s.web.servlet.DispatcherServlet : Completed 200 OK 2025-07-01T10:44:32.002-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.security.web.FilterChainProxy : Securing POST /fiberguardian/login 2025-07-01T10:44:32.002-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.s.w.a.c.ChannelProcessingFilter : Request: filter invocation [POST /fiberguardian/login]; ConfigAttributes: [REQUIRES\_SECURE\_CHANNEL] 2025-07-01T10:44:32.004-03:00 TRACE 4772 --- [fiberguardian] [io-8443-exec-10] s.s.w.c.CsrfTokenRequestAttributeHandler : Wrote a CSRF token to the following request attributes: [\_csrf, org.springframework.security.web.csrf.CsrfToken] 2025-07-01T10:44:32.004-03:00 TRACE 4772 --- [fiberguardian] [io-8443-exec-10] o.s.security.web.csrf.CsrfFilter : Found a CSRF token in the request 2025-07-01T10:44:32.005-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.security.web.csrf.CsrfFilter : Invalid CSRF token found for https://localhost:8443/fiberguardian/login 2025-07-01T10:44:32.005-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.s.w.access.AccessDeniedHandlerImpl : Responding with 403 status code 2025-07-01T10:44:32.012-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.security.web.FilterChainProxy : Securing POST /error 2025-07-01T10:44:32.012-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.s.w.a.c.ChannelProcessingFilter : Request: filter invocation [POST /error]; ConfigAttributes: [REQUIRES\_SECURE\_CHANNEL] 2025-07-01T10:44:32.012-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.s.w.a.AnonymousAuthenticationFilter : Set SecurityContextHolder to anonymous SecurityContext 2025-07-01T10:44:32.013-03:00 DEBUG 4772 --- [fiberguardian] [io-8443-exec-10] o.s.s.w.a.Http403ForbiddenEntryPoint : Pre-authenticated entry point called. Rejecting access

**O ChatGPT disse:**

Ótima observação. O fato de o controller POST /fiberguardian/login **nem ser chamado** confirma com alta certeza que:

**🚨 A requisição está bloqueando no filtro de CSRF, antes do Spring MVC entrar em ação.**

**🔁 Fluxo confirmado via log**

* GET /csrf-token:  
  ✅ CSRF gerado e associado a uma sessão — log mostra token escrito corretamente.
* POST /fiberguardian/login:  
  🔒 **Token CSRF encontrado**, **mas considerado inválido** → bloqueado por CsrfFilter  
  🔁 **Handler de acesso negado retorna 403 imediatamente**  
  🚫 O Spring **nem chama o controller** de login.

**✅ Confirmação do problema**

Isso confirma **sem dúvida** que a requisição POST /login está enviando o token CSRF **de uma sessão diferente da que foi usada para gerá-lo**.

Ou seja:

❌ O cookie JSESSIONID não está sendo enviado na requisição POST /login.

**✅ Passos para verificação imediata no navegador**

**1. Abra DevTools → aba Network**

Faça o login normalmente e observe:

**A. Requisição GET /csrf-token**

* Vá até **Headers** → **Request Headers**
* Veja se há:

makefile

CopiarEditar

Cookie: JSESSIONID=ABC123...

* Vá até **Response Headers**
  + Verifique Set-Cookie: JSESSIONID=...
  + Verifique Set-Cookie: XSRF-TOKEN=...

✅ Se sim, o servidor criou e enviou corretamente.

**B. Requisição POST /login**

* Em **Request Headers**, veja se **o cabeçalho Cookie está presente**:

makefile

CopiarEditar

Cookie: JSESSIONID=ABC123...

* Se estiver ausente, o problema está no **envio de cookies entre domínios HTTPS**.

**📌 Causa provável: navegador não envia cookie Secure porque o HTML foi carregado por http://localhost:8080**

Mesmo que sua API esteja em https://localhost:8443, **se a página HTML tiver sido servida por http://localhost:8080**, o navegador **nunca envia cookies com Secure=true**, mesmo com credentials: "include".

**✅ Solução recomendada**

**✅ 1. Sirva seu frontend via HTTPS também**

Configure seu servidor estático para servir os arquivos HTML em:

arduino

CopiarEditar

https://localhost:8080

Com certificado confiável ou self-signed aceito manualmente.

Exemplo: usando o Node.js com http-server e certificado:

bash

CopiarEditar

npx http-server ./public -p 8080 --ssl --cert cert.pem --key key.pem

Ou com NGINX / Apache configurado com TLS.

**✅ 2. Alternativamente, sirva tudo no mesmo domínio e porta (apenas para testes locais)**

Para facilitar em desenvolvimento, sirva o frontend estático **pelo próprio Spring Boot**, como recurso estático mapeado em /public/.

Exemplo:

* Mova os HTMLs para src/main/resources/static
* Acesse https://localhost:8443/login.html

Isso evita qualquer problema de origem ou de SameSite + Secure.

**🧪 Solução rápida para depurar**

Se quiser confirmar isso imediatamente:

1. Acesse manualmente https://localhost:8443/fiberguardian/csrf-token
2. Veja se o cookie JSESSIONID é setado.
3. Depois, abra o console JS do navegador e simule a chamada fetch('/fiberguardian/login', ...) com credentials: 'include'
4. Veja se o cookie JSESSIONID aparece na requisição.

**✅ Conclusão**

Você está com tudo muito bem configurado no backend. O problema está isolado no **mecanismo de cookie Secure/SameSite + origens diferentes**.

📌 O **backend está certo**.  
📌 O **frontend JS está certo**.  
✅ A **solução**: **garantir que o frontend seja servido via HTTPS** e da **mesma origem (ou ao menos mesma raiz de domínio)** para que o cookie JSESSIONID seja incluído.

Se quiser, posso te ajudar a montar um servidor local simples com HTTPS para servir os arquivos estáticos. Deseja isso?

**Você disse:**

vejo o cabeçalho da resposta de GET /csrf-token --- HTTP/1.1 200 Vary: Origin Vary: Access-Control-Request-Method Vary: Access-Control-Request-Headers Access-Control-Allow-Origin: https://127.0.0.1:8080 Access-Control-Allow-Credentials: true Set-Cookie: XSRF-TOKEN=08c99291-33de-431a-9c4d-1c14d8f9f41b; Path=/; Secure X-Content-Type-Options: nosniff X-XSS-Protection: 0 Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 Strict-Transport-Security: max-age=31536000 ; includeSubDomains X-Frame-Options: DENY Content-Type: application/json Transfer-Encoding: chunked Date: Tue, 01 Jul 2025 14:03:53 GMT Keep-Alive: timeout=60 Connection: keep-alive

**O ChatGPT disse:**

Excelente, Giovanni — com essa nova informação, o cenário está quase totalmente fechado para diagnóstico. Vamos por partes:

**📌 Análise do GET /csrf-token**

Você acessou via:

bash

CopiarEditar

https://localhost:8443/fiberguardian/csrf-token

E recebeu os headers:

yaml

CopiarEditar

Access-Control-Allow-Origin: https://127.0.0.1:8080

Access-Control-Allow-Credentials: true

Set-Cookie: XSRF-TOKEN=...; Path=/; Secure

**⚠️ Problema: o cookie JSESSIONID não foi setado**

O header Set-Cookie mostra **apenas o XSRF-TOKEN**, **não há JSESSIONID** na resposta.

Isso confirma que a **sessão não foi criada na chamada a /csrf-token**.

**✅ O que precisa acontecer:**

Para o token CSRF ser considerado válido no POST /login, ele precisa estar vinculado a **uma sessão**. Portanto:

* Na chamada GET /csrf-token, o servidor deve **criar uma sessão (JSESSIONID)**.
* Isso é feito chamando request.getSession(true) em algum ponto do fluxo da requisição.

**🛠️ Solução imediata: forçar criação da sessão no CsrfController**

**✅ Passo recomendado**

Verifique seu controller CsrfController, que trata /fiberguardian/csrf-token, e **adicione isso no início do método**:

java

CopiarEditar

@RequestMapping(value = "/csrf-token", method = RequestMethod.GET)

public Map<String, String> getCsrfToken(HttpServletRequest request) {

// ✅ Força criação da sessão para gerar JSESSIONID

HttpSession session = request.getSession(true);

CsrfToken csrfToken = (CsrfToken) request.getAttribute(CsrfToken.class.getName());

Map<String, String> responseBody = new HashMap<>();

responseBody.put("token", csrfToken.getToken());

responseBody.put("parameterName", csrfToken.getParameterName());

responseBody.put("headerName", csrfToken.getHeaderName());

return responseBody;

}

**✅ O que isso resolve**

Com request.getSession(true), o Spring cria a sessão e inclui o cookie:

mathematica

CopiarEditar

Set-Cookie: JSESSIONID=xyz123; Path=/; Secure; SameSite=Lax

Esse cookie é **enviado de volta** pelo navegador nas próximas requisições POST, como /login, **mantendo o vínculo com o token CSRF**, que já está correto.

**✅ Checklist de correções finais**

| **Item** | **Situação Esperada** |
| --- | --- |
| GET /csrf-token cria JSESSIONID | ✅ com request.getSession(true) |
| Resposta inclui dois cookies | Set-Cookie: JSESSIONID=... e XSRF-TOKEN=... |
| Frontend usa credentials: "include" | ✅ você já faz |
| Origem frontend é https://127.0.0.1:8080 | ✅ ou https://localhost:8080 com CORS |
| Navegador envia JSESSIONID e X-XSRF-TOKEN no POST /login | ✅ com isso o token será aceito |

**✅ Conclusão**

📌 **O backend não estava criando sessão no /csrf-token**, por isso o Spring não enviava JSESSIONID.

📎 A correção simples é **forçar a criação da sessão com request.getSession(true) no controller**.

Assim, seu fluxo de autenticação via REST com CSRF e SameSite=Lax continuará seguro e funcional.

Se quiser, posso revisar esse controller completo ou te ajudar a logar o valor do JSESSIONID para depuração no browser. Deseja?

**Você disse:**

entendi...o sessionid PRECISA ser construido já na requisicao /csrf-token eu pensei que o sessioin era construido DEPOIS da autentição. Realmente nos meus testes com curl, ao enviar a requisicao /csr-token só o token crsf era criado no cookie, o sessionid aparecei depois da requisicao de login

**O ChatGPT disse:**

Perfeito, Giovanni — seu raciocínio é claro e está **correto do ponto de vista conceitual comum**, mas no caso do Spring Security com CSRF, o comportamento necessário é um pouco **contraintuitivo**. Vamos explorar por que:

**🧠 Entendendo a Relação entre Sessão, CSRF e Spring Security**

**✅ 1. Spring Security associa o token CSRF à sessão**

* O CookieCsrfTokenRepository **armazena o token CSRF dentro da sessão HTTP do usuário**.
* Isso quer dizer que **sem sessão, não há "referência válida" de CSRF no backend**.

**🔄 2. O token CSRF é gerado *antes* da autenticação**

* O objetivo do CSRF é **proteger o processo de login** (e todas as requisições subsequentes) de ataques entre sites.
* Por isso, o token CSRF é exigido até mesmo na chamada ao endpoint /login, mesmo antes do usuário estar autenticado.

**⚠️ Seu erro inicial (muito comum e compreensível)**

Você imaginou que a sessão seria criada **após a autenticação bem-sucedida**, o que é **normal** para fluxos típicos.

Mas…

📌 **No caso de CSRF com CookieCsrfTokenRepository, a sessão precisa existir *antes* do login**, para que:

1. O GET /csrf-token gere um token
2. O backend **associe esse token à sessão**
3. O navegador receba o token + o JSESSIONID correspondente
4. No POST /login, o browser envie **ambos**
5. O Spring valide que **o token recebido pertence à sessão identificada pelo JSESSIONID**

**🧪 Comportamento típico que você viu com curl**

Ao fazer:

bash

CopiarEditar

curl -k https://localhost:8443/fiberguardian/csrf-token

Você recebe:

javascript

CopiarEditar

Set-Cookie: XSRF-TOKEN=abc123...

Mas não:

javascript

CopiarEditar

Set-Cookie: JSESSIONID=...

💡 Isso acontece porque o Spring só cria a sessão se ela for **explicitamente solicitada** com:

java

CopiarEditar

request.getSession(true)

Caso contrário, ele assume: "não precisa de sessão agora".

**✅ Melhor prática: criar sessão antecipadamente**

No controller /csrf-token, sempre chame:

java

CopiarEditar

request.getSession(true);

Isso:

* Garante que o JSESSIONID seja gerado e enviado
* Garante que o token CSRF fique **vinculado** à sessão correta

**✅ Resultado: fluxo seguro e funcional**

1. Frontend chama GET /csrf-token  
   🔒 → backend cria sessão + gera token CSRF  
   🍪 → navegador recebe cookies JSESSIONID e XSRF-TOKEN
2. Frontend faz POST /login  
   📤 → envia ambos os cookies e o token CSRF no header  
   ✅ → backend encontra a sessão correta e valida o token CSRF

**🧩 Dica extra: testando com curl o fluxo completo**

bash

CopiarEditar

# Obter o token e cookies (note o -c para salvar os cookies)

curl -k -c cookies.txt https://localhost:8443/fiberguardian/csrf-token

# Verificar o conteúdo dos cookies

cat cookies.txt

# Usar os cookies e token para autenticar

curl -k -b cookies.txt -X POST https://localhost:8443/fiberguardian/login \

-H "Content-Type: application/json" \

-H "X-XSRF-TOKEN: <coloque\_aqui\_o\_token>" \

-d '{"email": "admin@teste.com", "senha": "123456"}'

Se quiser, posso te ajudar a montar uma automação de testes com curl, Postman ou até testcontainers para validar isso em desenvolvimento local. Deseja?

**Você disse:**

vou te passar meu controller completo do get /csrf-token e voce implementa as alteracoes para gerar o sessionid? Segue: @GetMapping("/csrf-token") public ResponseEntity<Map<String, Object>> getCsrfToken(HttpServletRequest request) { try { CsrfToken token = (CsrfToken) request.getAttribute(CsrfToken.class.getName()); if (token == null || token.getToken() == null) { logger.warn("CSRF token não disponível na requisição"); return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR) .body(Map.of("error", "CSRF token não disponível")); } Map<String, Object> tokenInfo = new HashMap<>(); tokenInfo.put("token", token.getToken()); // token XORed tokenInfo.put("headerName", token.getHeaderName()); tokenInfo.put("parameterName", token.getParameterName()); tokenInfo.put("cookieName", "XSRF-TOKEN"); return ResponseEntity.ok(tokenInfo); } catch (Exception e) { logger.error("Erro ao obter token CSRF", e); return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR) .body(Map.of("error", "Erro ao obter token CSRF")); } }

**O ChatGPT disse:**

Claro, Giovanni. Abaixo está sua versão ajustada do controller GET /csrf-token, com a **inclusão explícita da criação de sessão via request.getSession(true)** para garantir que o backend envie o cookie JSESSIONID junto com o XSRF-TOKEN:

java

CopiarEditar

@GetMapping("/csrf-token")

public ResponseEntity<Map<String, Object>> getCsrfToken(HttpServletRequest request) {

try {

// ✅ Garante que a sessão seja criada (gera o cookie JSESSIONID)

HttpSession session = request.getSession(true);

CsrfToken token = (CsrfToken) request.getAttribute(CsrfToken.class.getName());

if (token == null || token.getToken() == null) {

logger.warn("CSRF token não disponível na requisição");

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)

.body(Map.of("error", "CSRF token não disponível"));

}

Map<String, Object> tokenInfo = new HashMap<>();

tokenInfo.put("token", token.getToken()); // token XORed

tokenInfo.put("headerName", token.getHeaderName());

tokenInfo.put("parameterName", token.getParameterName());

tokenInfo.put("cookieName", "XSRF-TOKEN");

// Opcional: Log de depuração

logger.debug("Sessão criada: {}", session.getId());

logger.debug("Token CSRF retornado: {}", token.getToken());

return ResponseEntity.ok(tokenInfo);

} catch (Exception e) {

logger.error("Erro ao obter token CSRF", e);

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)

.body(Map.of("error", "Erro ao obter token CSRF"));

}

}

**✅ O que isso garante:**

* A sessão será sempre criada quando esse endpoint for chamado;
* O Spring enviará automaticamente o Set-Cookie: JSESSIONID=...;
* O token CSRF estará vinculado a essa sessão;
* A próxima chamada POST /login poderá ser validada corretamente, desde que o navegador envie:
  + o cookie JSESSIONID correspondente;
  + o header X-XSRF-TOKEN.

Se quiser posso te ajudar a instrumentar testes no navegador ou via curl para verificar o Set-Cookie e garantir que o fluxo está completo. Deseja isso também?

**Você disse:**

e no controller do login, ele não vai gerar um novo session id? Veja o codigo: @PostMapping("/login") public ResponseEntity<?> login(@RequestBody @Validated UsuarioAutenticadoInput loginRequest, HttpServletRequest request) { System.out.println("JSESSIONID recebido: " + request.getSession(false).getId()); // Log para depuração System.out.println("Token CSRF esperado: " + request.getAttribute("\_csrf")); // Log para depuração UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(loginRequest.getEmail(), loginRequest.getSenha()); try { Authentication authentication = authenticationManager.authenticate(authToken); SecurityContextHolder.getContext().setAuthentication(authentication); // Garante criação da sessão // Garante criação da sessão HttpSession session = request.getSession(true); System.out.println("Nova sessão criada: " + session.getId()); // Log para depuração UsuarioAutenticado usuarioAutenticado = (UsuarioAutenticado) authentication.getPrincipal(); Usuario usuario = usuarioAutenticado.getUsuario(); UsuarioLoginResponseDto responseDto = modelMapper.map(usuario, UsuarioLoginResponseDto.class); return ResponseEntity.ok(responseDto); } catch (AuthenticationException ex) { return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Credenciais inválidas"); } }

**O ChatGPT disse:**

Excelente ponto, Giovanni. Vamos analisar com precisão o comportamento da sessão no seu controller de login e esclarecer **o que o request.getSession(true) está realmente fazendo** nesse contexto.

**✅ O que já sabemos:**

* No GET /csrf-token, você agora **cria a sessão antecipadamente**, o que garante o envio de JSESSIONID e vínculo do token CSRF com essa sessão.
* O cliente envia de volta:
  + o JSESSIONID via Cookie
  + o token CSRF via X-XSRF-TOKEN no header
* O filtro CsrfFilter valida a combinação antes de qualquer controller ser executado.

**🔍 No seu POST /login**

Trecho relevante:

java

CopiarEditar

System.out.println("JSESSIONID recebido: " + request.getSession(false).getId()); // Log para depuração

...

HttpSession session = request.getSession(true);

System.out.println("Nova sessão criada: " + session.getId());

**💡 O que isso significa:**

* request.getSession(false)  
  → retorna a sessão **existente** vinculada ao JSESSIONID recebido do cliente (se houver).
* request.getSession(true)  
  → retorna a sessão existente **ou cria uma nova** se não existir.

🔍 Mas no seu caso, a sessão **já existe**, pois ela foi criada no GET /csrf-token. Então:

⚠️ **Você não precisa (e não deve) chamar getSession(true) de novo no POST /login se quiser manter a mesma sessão.**

**✅ Melhor abordagem para reaproveitar a sessão**

**🛠 Substitua este trecho:**

java

CopiarEditar

// Garante criação da sessão

HttpSession session = request.getSession(true);

System.out.println("Nova sessão criada: " + session.getId());

**Por este:**

java

CopiarEditar

HttpSession session = request.getSession(false);

if (session == null) {

// Segurança extra: recusar login se não houver sessão válida (token CSRF foi inválido)

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Sessão inválida ou expirada");

}

System.out.println("Reutilizando sessão existente: " + session.getId());

**✅ Justificativa técnica**

O Spring Security **valida o token CSRF antes de chamar seu controller**, e essa validação só passa se:

* o token for válido
* e estiver associado à mesma sessão JSESSIONID

Se você criar **uma nova sessão após o login**, isso **quebraria a continuidade** da autenticação e **perderia a associação CSRF + JSESSIONID** — o que pode causar confusão, especialmente se houver redirecionamento ou APIs protegidas logo após o login.

**✅ Conclusão**

📌 **Não recrie a sessão no /login**.  
📌 **Reutilize a sessão já criada no /csrf-token**.  
📌 Isso mantém a integridade da sessão e da validação CSRF.

Se você quiser aplicar medidas adicionais como sessionFixation().migrateSession() (que você já tem no SecurityFilterChain), o Spring vai cuidar automaticamente de gerar um novo ID **seguro** pós-login, sem comprometer o fluxo CSRF.

Se quiser, posso ajustar o controller completo com essa correção. Deseja isso?