Entendido! Sua configuração do Spring Security e sua ideia de enviar o token CSRF \*\*uma única vez\*\* para o frontend, armazená-lo em memória no frontend, e usá-lo em todas as requisições subsequentes (inclusive a de login) fazem sentido e são uma abordagem válida. Vou analisar sua configuração, explicar como implementar sua ideia com sua página de login customizada (estática HTML), e ajustar o frontend para armazenar o token CSRF em memória, eliminando a necessidade de múltiplas requisições para obter o token. Também avaliarei se essa abordagem é uma boa prática e responderei diretamente à sua preocupação.

---

### Análise da Configuração do Spring Security

Sua configuração do `SecurityConfig` está bem estruturada e suporta o uso do cookie `XSRF-TOKEN` com o `CookieCsrfTokenRepository`. Vamos revisar os pontos principais e como eles se alinham com sua ideia:

1. \*\*CSRF Configuration\*\*:

- Você usa `CookieCsrfTokenRepository.withHttpOnlyFalse()`, que envia o token CSRF como cookie `XSRF-TOKEN`, acessível pelo JavaScript (devido ao `HttpOnly=false`).

- O `XorCsrfTokenRequestAttributeHandler` é configurado para lidar com tokens CSRF de forma segura, usando XOR para validação.

- O endpoint `/fiberguardian/csrf-token` está isento de proteção CSRF (`.ignoringRequestMatchers("/fiberguardian/csrf-token")`), permitindo que o frontend obtenha o token sem autenticação.

2. \*\*Login e Autorização\*\*:

- O endpoint `/fiberguardian/login` (POST) é permitido sem autenticação (`.permitAll()`), o que é correto para sua página de login customizada.

- O `formLogin` está desativado (`.formLogin(form -> form.disable())`), confirmando que você não usa o formulário padrão do Spring Security.

- A proteção CSRF está ativa para o endpoint `/fiberguardian/login`, exigindo o header `X-XSRF-TOKEN` nas requisições POST.

3. \*\*Sessão\*\*:

- A política de sessão `SessionCreationPolicy.IF\_REQUIRED` cria uma sessão apenas quando necessário, com `migrateSession()` para proteção contra fixação de sessão.

- O cookie `JSESSIONID` e `XSRF-TOKEN` são gerenciados automaticamente pelo navegador.

4. \*\*Sua Ideia\*\*:

- Você quer que o frontend obtenha o token CSRF \*\*uma vez\*\* (via endpoint `/fiberguardian/csrf-token` ou cookie `XSRF-TOKEN`) e armazene-o em memória (ex.: variável JavaScript) para uso em todas as requisições subsequentes, incluindo o login.

- Você considera desnecessário o backend enviar o token CSRF a cada requisição, já que o token pode ser reutilizado enquanto a sessão for válida.

\*\*Boa Prática?\*\*

- \*\*Armazenar o token CSRF em memória\*\*: Essa abordagem é válida e eficiente, desde que o token CSRF seja válido para toda a sessão (o que é o caso com `CookieCsrfTokenRepository` e `setCookieMaxAge(-1)` para cookie de sessão). Armazenar o token em memória no frontend reduz requisições desnecessárias e mantém o fluxo simples.

- \*\*Evitar múltiplos envios do token\*\*: Você está correto que não há necessidade de o backend enviar o token CSRF a cada requisição. O Spring Security gera um token CSRF por sessão, e ele permanece válido até a sessão expirar ou ser invalidada (ex.: logout). Armazenar o token no frontend alinha-se com essa lógica.

- \*\*Segurança\*\*: O uso do `XorCsrfTokenRequestAttributeHandler` e do header `X-XSRF-TOKEN` garante proteção contra ataques CSRF. Armazenar o token em memória no JavaScript é seguro, desde que a aplicação use HTTPS (como configurado com `.requiresChannel(channel -> channel.anyRequest().requiresSecure())`).

- \*\*Melhorias sugeridas\*\*:

- Como você já usa `CookieCsrfTokenRepository`, o token CSRF pode ser obtido diretamente do cookie `XSRF-TOKEN` ao carregar a página, eliminando a necessidade do endpoint `/fiberguardian/csrf-token`. Isso reduz uma requisição e simplifica o fluxo.

- Adicionar feedback visual no frontend (ex.: mensagens de erro, estado de carregamento).

- Garantir que o frontend lida com cenários onde o token CSRF não está disponível (ex.: cookie bloqueado).

---

### Implementação da Sua Ideia

Vou ajustar o frontend para:

1. Obter o token CSRF \*\*uma vez\*\* ao carregar a página (usando o cookie `XSRF-TOKEN` ou, opcionalmente, o endpoint `/fiberguardian/csrf-token`).

2. Armazenar o token em memória (variável JavaScript).

3. Usar o token armazenado em todas as requisições, incluindo a de login.

4. Manter sua página de login customizada (estática HTML).

#### 1. Configuração do Backend

Sua configuração do `SecurityConfig` já está correta para enviar o cookie `XSRF-TOKEN` automaticamente ao carregar a página de login. Como você prefere obter o token \*\*uma vez\*\* e armazená-lo, podemos aproveitar o cookie `XSRF-TOKEN` gerado pelo `CookieCsrfTokenRepository`. Assim, você pode \*\*remover o endpoint `/fiberguardian/csrf-token`\*\* do seu controller, pois o cookie já contém o token necessário.

\*\*Ajustes no `SecurityConfig`\*\*:

- Remova apenas a isenção do endpoint `/fiberguardian/csrf-token`, já que ele não será mais necessário.

- Mantenha o restante da configuração como está.

```x-java

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.logout.HttpStatusReturningServerLogoutSuccessHandler;

import org.springframework.security.web.csrf.CookieCsrfTokenRepository;

import org.springframework.security.web.csrf.XorCsrfTokenRequestAttributeHandler;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration

public class SecurityConfig {

@Bean

AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

@Bean

XorCsrfTokenRequestAttributeHandler xorCsrfTokenRequestAttributeHandler() {

return new XorCsrfTokenRequestAttributeHandler();

}

@Bean

CsrfTokenRepository csrfTokenRepository() {

CookieCsrfTokenRepository repository = CookieCsrfTokenRepository.withHttpOnlyFalse();

repository.setCookieName("XSRF-TOKEN");

repository.setHeaderName("X-XSRF-TOKEN");

repository.setParameterName("\_csrf");

repository.setCookiePath("/");

repository.setCookieMaxAge(-1); // Session cookie

repository.setSecure(true); // HTTPS only

return repository;

}

@Bean

SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

// Configura CORS para frontend estático

.cors(cors -> cors.configurationSource(corsConfigurationSource()))

// Configura CSRF com cookie acessível pelo frontend

.addFilterAfter(new SameSiteCookieFilter(), CsrfFilter.class)

.csrf(csrf -> csrf

.csrfTokenRepository(csrfTokenRepository())

.csrfTokenRequestHandler(new XorCsrfTokenRequestAttributeHandler())

)

// Força HTTPS

.requiresChannel(channel -> channel.anyRequest().requiresSecure())

// Configura autorização

.authorizeHttpRequests(authz -> authz

.requestMatchers(HttpMethod.POST, "/fiberguardian/login").permitAll()

.requestMatchers(HttpMethod.GET, "/public/\*\*", "/login").permitAll()

.anyRequest().authenticated()

)

// Configura sessões

.sessionManagement(session -> session

.sessionCreationPolicy(SessionCreationPolicy.IF\_REQUIRED)

.sessionFixation().migrateSession()

.maximumSessions(1)

.maxSessionsPreventsLogin(false)

)

// Desativa form login

.formLogin(form -> form.disable())

// Configura logout

.logout(logout -> logout

.logoutUrl("/fiberguardian/logout")

.logoutSuccessHandler(new HttpStatusReturningServerLogoutSuccessHandler())

.invalidateHttpSession(true)

.deleteCookies("JSESSIONID", "XSRF-TOKEN")

);

return http.build();

}

// Configuração de CORS (exemplo, ajuste conforme necessário)

private CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(List.of("http://localhost:8080")); // Ajuste para o domínio do frontend

configuration.setAllowedMethods(List.of("GET", "POST"));

configuration.setAllowCredentials(true);

configuration.setAllowedHeaders(List.of("Content-Type", "X-XSRF-TOKEN"));

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/fiberguardian/\*\*", configuration);

return source;

}

}

```

\*\*Mudanças\*\*:

- Removida a linha `.ignoringRequestMatchers("/fiberguardian/csrf-token")`, pois o endpoint não será mais usado.

- Adicionada permissão para `/login` (GET) em `.requestMatchers(HttpMethod.GET, "/public/\*\*", "/login").permitAll()` para garantir que a página estática seja acessível.

- Adicionado um exemplo de `corsConfigurationSource` para suportar CORS, caso o frontend esteja em um domínio diferente (ex.: `http://localhost:8080`).

\*\*Remover o Endpoint `/csrf-token`\*\*:

Como o `CookieCsrfTokenRepository` já envia o cookie `XSRF-TOKEN` automaticamente, você pode remover o método `@GetMapping("/csrf-token")` do seu controller:

```java

// Remova este método do controller

@GetMapping("/csrf-token")

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

// ... código atual ...

}

```

\*\*Servindo a Página Estática\*\*:

Certifique-se de que `login.html` está em `src/main/resources/static/` e mapeado para `/login`. Crie um controlador simples, se necessário:

```x-java

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class LoginController {

@GetMapping("/login")

public String showLoginPage() {

return "login"; // Refere-se a src/main/resources/static/login.html

}

}

```

Alternativamente, se você acessar a página diretamente como `/login.html`, ajuste a permissão no `SecurityConfig`:

```java

.requestMatchers(HttpMethod.GET, "/public/\*\*", "/login.html").permitAll()

```

---

#### 2. Ajustes no Frontend

O frontend será modificado para:

- Ler o cookie `XSRF-TOKEN` \*\*uma vez\*\* ao carregar a página e armazená-lo em uma variável global no namespace.

- Usar o token armazenado em memória para todas as requisições, incluindo a de login.

- Manter a página de login customizada (`login.html`).

\*\*HTML (`login.html`)\*\*:

O HTML permanece idêntico ao que você forneceu, com a inclusão do `login.js`. Como não há mudanças na estrutura, usarei o mesmo código:

```html

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<title>Login - FiberGuardian</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">

<link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css" rel="stylesheet">

</head>

<body class="bg-light">

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-4">

<div class="card shadow-lg rounded-4">

<div class="card-body text-center">

<!-- Logo SVG centralizado -->

<div class="mb-4">

<div class="d-flex justify-content-center">

<svg viewBox="0 0 1024 1024" preserveAspectRatio="xMidYMid meet" style="width: 100%; height: auto; max-height: 350px;" xmlns="http://www.w3.org/2000/svg">

<!-- SVG removido para brevidade, mantenha o original -->

</svg>

</div>

</div>

<form method="POST" action="/login">

<div class="mb-3 text-start">

<label for="email" class="form-label">E-mail</label>

<input type="email" class="form-control" id="email" name="email" required autofocus>

</div>

<div class="mb-3 text-start">

<label for="senha" class="form-label">Senha</label>

<input type="password" class="form-control" id="senha" name="senha" required>

</div>

<div class="d-grid">

<button type="submit" class="btn btn-primary">

<i class="fa-solid fa-right-to-bracket me-1"></i> Entrar

</button>

</div>

</form>

<div class="text-center mt-3">

<a href="tela\_cadastro.html" class="text-decoration-none">

<i class="fa-solid fa-user-plus me-1"></i> Novo por aqui? Cadastre-se

</a>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="js/login.js"></script>

</body>

</html>

```

\*\*JavaScript (`login.js`)\*\*:

O script armazena o token CSRF em uma variável no namespace `FiberGuardian.Auth` ao carregar a página e o reutiliza em todas as requisições.

```javascript

// Namespace para o módulo de login

const FiberGuardian = FiberGuardian || {};

// Módulo de autenticação

FiberGuardian.Auth = (function () {

// Configurações

const API\_BASE\_URL = 'https://localhost:8443/fiberguardian';

const LOGIN\_ENDPOINT = `${API\_BASE\_URL}/login`;

// Armazena o token CSRF em memória

let csrfToken = null;

// Função para obter o valor de um cookie

function getCookie(name) {

const value = `; ${document.cookie}`;

const parts = value.split(`; ${name}=`);

if (parts.length === 2) return parts.pop().split(';').shift();

return null;

}

// Função para inicializar o token CSRF

function initCsrfToken() {

csrfToken = getCookie('XSRF-TOKEN');

if (!csrfToken) {

console.warn('Token CSRF não encontrado. Tente recarregar a página.');

}

}

// Função para realizar o login

async function login(email, senha) {

try {

// Verifica se o token CSRF está disponível

if (!csrfToken) {

throw new Error('Token CSRF não encontrado. Tente recarregar a página.');

}

// Envia a requisição de login

const response = await fetch(LOGIN\_ENDPOINT, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'X-XSRF-TOKEN': csrfToken,

},

credentials: 'include', // Inclui cookies (JSESSIONID, XSRF-TOKEN)

body: JSON.stringify({ email, senha }),

});

if (!response.ok) {

const errorData = await response.json();

if (response.status === 401) {

throw new Error('Email ou senha inválidos');

} else if (response.status === 403) {

throw new Error('Erro de segurança (CSRF inválido). Tente novamente.');

} else {

throw new Error(errorData.message || 'Erro ao realizar login');

}

}

// Login bem-sucedido, redireciona ou atualiza a UI

return await response.json(); // Supõe que o backend retorna dados úteis

} catch (error) {

console.error('Erro no login:', error);

throw error;

}

}

// Função para exibir mensagens de erro

function showError(message) {

const errorDiv = document.createElement('div');

errorDiv.className = 'alert alert-danger mt-3';

errorDiv.textContent = message;

const form = document.querySelector('form');

form.parentNode.insertBefore(errorDiv, form.nextSibling);

// Remove a mensagem após 5 segundos

setTimeout(() => errorDiv.remove(), 5000);

}

// Função para gerenciar o estado de carregamento do botão

function toggleButtonLoading(button, isLoading) {

if (isLoading) {

button.disabled = true;

button.innerHTML = '<i class="fa-solid fa-spinner fa-spin me-1"></i> Entrando...';

} else {

button.disabled = false;

button.innerHTML = '<i class="fa-solid fa-right-to-bracket me-1"></i> Entrar';

}

}

// Inicialização do módulo

function init() {

// Inicializa o token CSRF ao carregar a página

initCsrfToken();

const form = document.querySelector('form');

if (!form) {

console.error('Formulário de login não encontrado');

return;

}

form.addEventListener('submit', async (event) => {

event.preventDefault(); // Impede o envio padrão do formulário

// Remove mensagens de erro anteriores

const existingError = document.querySelector('.alert');

if (existingError) existingError.remove();

const email = document.getElementById('email').value.trim();

const senha = document.getElementById('senha').value.trim();

const button = form.querySelector('button');

// Validação simples no frontend

if (!email || !senha) {

showError('Por favor, preencha todos os campos.');

return;

}

toggleButtonLoading(button, true);

try {

const result = await login(email, senha);

// Redireciona para a página principal após login bem-sucedido

window.location.href = '/dashboard.html'; // Ajuste conforme necessário

} catch (error) {

showError(error.message || 'Erro ao realizar login. Tente novamente.');

} finally {

toggleButtonLoading(button, false);

}

});

}

// Métodos públicos

return {

init,

};

})();

// Inicializa o módulo quando o DOM estiver carregado

document.addEventListener('DOMContentLoaded', FiberGuardian.Auth.init);

```

\*\*Mudanças no JavaScript\*\*:

- \*\*Armazenamento em Memória\*\*: O token CSRF é armazenado na variável `csrfToken` no namespace `FiberGuardian.Auth`. Ele é inicializado uma vez na função `initCsrfToken`, chamada durante a inicialização do módulo.

- \*\*Leitura do Cookie\*\*: O token é obtido do cookie `XSRF-TOKEN` ao carregar a página, usando a função `getCookie`.

- \*\*Reutilização\*\*: A função `login` usa o `csrfToken` armazenado em memória, sem fazer novas requisições para obter o token.

- \*\*Validação\*\*: Se o token não estiver disponível, exibe um erro pedindo para recarregar a página.

- \*\*Feedback Visual\*\*: Mantém o estado de carregamento no botão e mensagens de erro com Bootstrap.

---

### Fluxo Final

1. O usuário acessa a página de login estática (`/login` ou `/login.html`).

2. O Spring Security gera um token CSRF e o envia como cookie `XSRF-TOKEN` automaticamente (devido ao `CookieCsrfTokenRepository`).

3. O JavaScript lê o cookie `XSRF-TOKEN` ao carregar a página e armazena o token na variável `csrfToken`.

4. Ao clicar em "Entrar", o JavaScript envia uma requisição POST para `/fiberguardian/login` com:

- Header `X-XSRF-TOKEN` contendo o valor de `csrfToken`.

- Corpo JSON com `email` e `senha`.

- Cookies (`JSESSIONID`, `XSRF-TOKEN`) incluídos via `credentials: 'include'`.

5. O backend valida o token CSRF, autentica o usuário e atualiza o `JSESSIONID`.

6. Em caso de sucesso, o frontend redireciona para `/dashboard.html`.

---

### Resposta à Sua Ideia

> \*\*Minha ideia básica era enviar o token CSRF UMA vez para o frontend e todas as requisições (inclusive a de login) devem ter esse token (imagino que o frontend armazene) em memória esse token.\*\*

Sua ideia está implementada corretamente:

- O token CSRF é obtido \*\*uma vez\*\* do cookie `XSRF-TOKEN` ao carregar a página e armazenado na variável `csrfToken` no namespace `FiberGuardian.Auth`.

- Todas as requisições (incluindo a de login) usam esse token armazenado, sem necessidade de novas requisições ao backend.

- O cookie `XSRF-TOKEN` é gerado automaticamente pelo Spring Security, eliminando a necessidade do endpoint `/fiberguardian/csrf-token`.

> \*\*Não vejo sentido em o backend ficar enviando o token novamente a cada requisição.\*\*

Você está correto. O token CSRF gerado pelo `CookieCsrfTokenRepository` é válido para toda a sessão (devido ao `setCookieMaxAge(-1)`), então não há necessidade de enviá-lo repetidamente. Armazenar o token em memória no frontend é eficiente e alinha-se com o comportamento do Spring Security.

---

### Boas Práticas e Segurança

- \*\*Eficiência\*\*: Armazenar o token CSRF em memória elimina requisições adicionais, reduzindo a latência.

- \*\*Segurança\*\*:

- O uso do `XorCsrfTokenRequestAttributeHandler` e do header `X-XSRF-TOKEN` garante proteção robusta contra CSRF.

- O cookie `XSRF-TOKEN` é acessível pelo JavaScript (`HttpOnly=false`), mas isso é necessário para o frontend lê-lo. O `JSESSIONID` deve permanecer `HttpOnly` para segurança.

- A configuração `setSecure(true)` garante que os cookies sejam enviados apenas via HTTPS.

- \*\*Validação\*\*: O JavaScript verifica se o token está disponível antes de enviar a requisição, exibindo um erro amigável se necessário.

- \*\*Manutenção\*\*: Remover o endpoint `/csrf-token` simplifica o backend, mas você pode mantê-lo como fallback, se desejar:

```javascript

// Fallback opcional no JavaScript

async function initCsrfToken() {

csrfToken = getCookie('XSRF-TOKEN');

if (!csrfToken) {

const response = await fetch(`${API\_BASE\_URL}/csrf-token`, {

method: 'GET',

credentials: 'include',

});

if (!response.ok) throw new Error('Erro ao obter o token CSRF');

const data = await response.json();

csrfToken = data.token;

}

}

```

---

### Configurações Adicionais

1. \*\*Testes\*\*:

- Verifique na aba "Network" e "Application > Cookies" do navegador se o cookie `XSRF-TOKEN` é enviado ao carregar `/login`.

- Confirme que a requisição POST para `/fiberguardian/login` inclui o header `X-XSRF-TOKEN` com o valor correto.

- Teste casos de erro (ex.: credenciais inválidas, cookie ausente) para garantir que as mensagens de erro são exibidas.

2. \*\*Acessibilidade\*\*:

- Adicione atributos ARIA ao formulário para melhorar a acessibilidade:

```html

<form method="POST" action="/login" aria-labelledby="login-title">

<h2 id="login-title" class="visually-hidden">Formulário de Login</h2>

<!-- ... resto do formulário ... -->

</form>

```

3. \*\*CORS\*\*:

- Sua configuração de CORS já está incluída no `SecurityConfig`. Certifique-se de que o domínio do frontend (ex.: `http://localhost:8080`) está correto em `corsConfigurationSource`.

---

### Resumo

- \*\*Página Customizada\*\*: Sua página de login estática (`login.html`) é mantida, sem dependência do formulário padrão do Spring Security.

- \*\*Token CSRF Uma Vez\*\*: O token CSRF é obtido do cookie `XSRF-TOKEN` ao carregar a página e armazenado em memória (`csrfToken`) para uso em todas as requisições.

- \*\*Backend\*\*: O `CookieCsrfTokenRepository` envia o cookie `XSRF-TOKEN` automaticamente, permitindo remover o endpoint `/fiberguardian/csrf-token`.

- \*\*Boa Prática\*\*: Armazenar o token em memória é eficiente e seguro, alinhado com o comportamento do Spring Security (token válido por sessão).

Se precisar de ajustes, como integrar o fallback do endpoint `/csrf-token`, adicionar mais validações, ou configurar outros endpoints protegidos por CSRF, por favor, me avise!