Yes, you **should** use a CSRF protection mechanism if your JWT-based authentication system stores tokens in **cookies**.

In 2026, security standards continue to recommend a defense-in-depth approach because relying solely on browser-level flags like `SameSite` may not provide complete protection against all cross-site threats.

When CSRF Protection is Required

- **JWT in Cookies:** If you store your access or refresh tokens in cookies (especially `HttpOnly` cookies to prevent XSS), the browser automatically attaches them to cross-site requests, making your application vulnerable to CSRF.
- **Cross-Domain Architectures:** If your frontend and backend are on different domains (e.g., `SameSite=None`), CSRF tokens are strictly necessary because `SameSite` protections are either relaxed or disabled.

Recommended Implementation Strategies

1. **Double Submit Cookie Pattern:**
    1. **Stateless Mitigation:** The server sends a random CSRF token in a non-`HttpOnly` cookie.
    2. **Client Header:** Your frontend JavaScript reads this cookie and includes the value in a custom HTTP header (e.g., `X-XSRF-TOKEN`) for every state-changing request.
    3. **Verification:** The server verifies that the header value matches the cookie value. For maximum security, use the **Signed Double Submit Cookie** pattern to prevent cookie injection.
2. **SameSite Cookie Attribute:**
    1. Set cookies to `SameSite=Strict` or `SameSite=Lax`. While this is a powerful browser-level defense, it is best used alongside a CSRF token for defense-in-depth.
3. **Custom Request Headers:**
    1. Requiring a custom header for all API calls can mitigate CSRF, as malicious cross-site forms cannot easily set custom headers without preflight (CORS) approval.

When You Can Skip CSRF Tokens

- **Authorization Header:** If you store your JWT in **memory** or **localStorage** and manually attach it to the `Authorization: Bearer <token>` header, you are inherently protected from CSRF because browsers do not automatically attach these headers to forged requests. However, this method increases your risk of **XSS attacks**.

2026 Best Practices Checklist

- **Use HttpOnly and Secure:** Always set these flags for JWT cookies to prevent script access and ensure transmission over HTTPS.
- **Backend for Frontend (BFF):** For high-security applications, consider the BFF pattern where the browser never sees the access token directly, and all auth logic is handled by a server-side proxy using secure sessions.
- **Short Expirations:** Keep access tokens short-lived and use the refresh token to maintain the session.