ChatGPT

Armazenamento e Fluxo de Autenticação JWT

Para começar, **não armazene tokens JWT sensíveis em** `localStorage` **ou** `sessionStorage`, pois estes podem ser facilmente lidos por código JavaScript malicioso em caso de XSS [1] [2]. Em vez disso, pratique separar o **access token** (curto prazo) do **refresh token** (longo prazo). Por exemplo, uma abordagem recomendada é manter o *access token* apenas na memória/estado da aplicação (ou contexto React) e guardar o *refresh token* em um cookie seguro (`HttpOnly`, `Secure`, com `SameSite`) [1] [3]. Assim, mesmo que haja XSS, o token de atualização não fica exposto ao JavaScript. Ao carregar a página, sua app pode fazer uma requisição automática ao backend usando o refresh token armazenado no cookie para obter um novo access token e populá-lo no estado da aplicação (usando React Context, Zustand etc.).

Por exemplo, no próprio Medium de Ryan Chenkie, recomenda-se fortemente *"não armazenar tokens em local storage"* e, em vez disso, usar a memória da aplicação ou cookies HttpOnly [2] [1]. De fato, o uso de cookies HttpOnly para o refresh token protege contra scripts maliciosos, e manter o access token somente em memória evita que ele seja persistido em ambiente vulnerável. Assim, remova trechos que fazem `localStorage.setItem('jwt_token', token)` no frontend; passe a depender do cookie de sessão. Também configure o cookie com as flags apropriadas (`secure: true`, `httpOnly: true`, e idealmente `SameSite`) para fortalecer contra CSRF/XSS [3].

Axios: Headers, Credenciais e Interceptadores

Ao chamar a API, use **axios com** `withCredentials: true` para que cookies (incluindo o refresh token HttpOnly) sejam enviados em cada requisição cross-site [4]. Por exemplo:

```
axios.defaults.withCredentials = true;
```

Isso garante que o cookie de autenticação gerado pelo servidor seja enviado automaticamente sem expor o token ao JS. (O StackOverflow observa que `{ withCredentials: true }` envia credenciais se o CORS estiver configurado corretamente [4].) Além disso, você deve implementar um **interceptor de resposta** do axios para lidar com respostas 401 (token expirado). A estratégia típica é: ao receber 401, usar o refresh token para obter um novo access token e *retry* da requisição original [5] [6]. Em código, fica algo como:

```
axios.interceptors.response.use(
  res => res,
  async err => {
    if (err.response.status === 401 && !err.config._retry) {
      err.config._retry = true;
      const refreshRes = await axios.post('/refresh-token');
      const newToken = refreshRes.data.token;
      // atualizar storage com newToken se aplicável
      err.config.headers['Authorization'] = 'Bearer ' + newToken;
      return axios(err.config);
```

```
    }
    return Promise.reject(err);
  }
);
```

Esse padrão (destacado em exemplos online [5] [6]) garante que, se o access token expirar, o cliente automaticamente tenta renová-lo com o servidor, mantendo o usuário logado sem redirecionar ao login.

Estado Global de Autenticação (Zustand, Context)

Para refletir o estado do usuário logado na UI, use um gerenciador de estado como **Zustand**. Mas atenção: não armazene tokens JWT diretamente em `localStorage` dentro da store, pois isso persiste o token de forma insegura. Uma abordagem híbrida é ter o cookie HttpOnly como fonte da verdade e usar a store apenas para informações derivadas (por exemplo, decodificar o access token em memória e guardar `user` e flags de login) [1] [7]. Como sugere um artigo de autenticação com Zustand, o ideal é manter os tokens em cookies persistentes e sincronizar a store reativa com esses valores [7]. Por exemplo, ao inicializar o app você pode usar `zustand` para ler o cookie (ou fazer uma requisição de refresh) e então definir `accessToken` e dados do usuário no estado. Assim, componentes React podem reagir às mudanças de login.

Em suma, *unifique* a lógica de autenticação: use um único axios instanciado com as configurações (baseURL, withCredentials, interceptors) e uma única fonte de estado (por exemplo, sua store do Zustand ou Context) para `isAuthenticated`, `user`, etc. Isso evita duplicação e incongruências. O exemplo de doichevkostia.dev ilustra bem essa ideia: “tínhamos muitos lugares com dados de autenticação e nada estava normalizado. A solução ideal foi ter armazenamento persistente (cookies) e armazenamento interno (Zustand) que pudéssemos subscrever” [7].

Rotas Express e Middleware de Autenticação

No lado servidor, verifique que as rotas estão consistentes. Por exemplo, sua API registra rotas como `/links/:key`, `/protected/:key`, `/private/:key`, `/r/:key` etc. Cada rota deve corresponder às chamadas do frontend. O middleware `authenticateToken` deve procurar o JWT **ou no header Authorization** (Bearer) ou nos cookies configurados (como `token` ou `accessToken`) e validar com `jwt.verify`. O código fornecido já faz isso parcialmente. Garanta que o **token enviado pelo cliente** (seja via cookie ou header) combina com o método escolhido: *se você migrar para cookie HttpOnly, não precisará mais chamar* `axios.post` *com Authorization; basta usar* `withCredentials: true` *e deixar o browser enviar o cookie*. Em resumo, padronize: **ou tudo via cookie**, ou **tudo via header**, mas não misture sem necessidade.

Além disso, evite redirecionamentos abertos. Se sua lógica faz `res.redirect(link.url)`, assegure-se de que `link.url` seja um destino confiável (por exemplo, sempre um domínio interno ou pegue da base de dados) e não algo vindo do usuário sem validação [8]. Segundo o guia de segurança do Express, aplicações devem validar URLs antes de redirecionar para evitar *open redirects* (que podem levar usuários a sites maliciosos) [8]. Por exemplo, verifique se `link.url` corresponde ao domínio esperado antes de usar `res.redirect(link.url)`.

Por fim, configure **cookies de sessão com as flags apropriadas**. Como recomendado pelo Express, use cookies com `secure: true` (HTTPS only) e `httpOnly: true` (não acessível via JS) [3]. Se usar `SameSite=Lax` ou `Strict`, ajuda a mitigar CSRF. Essas boas práticas garantem que, mesmo que seu sistema de links redirecione, ele o faça de forma segura, sem exposição de tokens ou sessões.

Resumo das Correções

1. **Remova uso de `localStorage` para JWT.** Dependa de cookies seguros e / ou estado em memória.
2. **Use axios com `withCredentials: true`** para cookies e configure interceptors para refresh de token [4] [5].
3. **Valide variáveis de ambiente e rotas:** garanta que `VITE_ROTA_*` batem com as rotas do servidor.
4. **Padronize sua estratégia de autenticação:** defina se os tokens serão passados via cookie ou header e ajuste frontend/servidor de acordo.
5. **Reforce segurança no backend:** use middleware de autenticação corretamente (token ou cookie), proteja redirecionamentos e configure cookies com `httpOnly` e `secure` [3] [8].

Com essas mudanças – alinhadas às recomendações de segurança e fluxo de autenticação (ex. usar cookies HttpOnly para refresh e axios interceptors para renovação automática) [1] [5] – seu frontend conseguirá armazenar e renovar tokens corretamente e as funções de gerenciamento de links irão operar sem recursos inexistentes ou inconsistentes.

Fontes: Práticas recomendadas de armazenamento seguro de JWT [1] [2], uso de interceptors no Axios para refresh token [5] [6], e diretrizes de segurança do Express (open redirect, cookies seguras) [8] [3].

---

[1] security - Should JWT be stored in localStorage or cookie? - Stack Overflow
https://stackoverflow.com/questions/34817617/should-jwt-be-stored-in-localstorage-or-cookie

[2] React Authentication: How to Store JWT in a Cookie | by Ryan Chenkie | Medium
https://medium.com/@ryanchenkie_40935/react-authentication-how-to-store-jwt-in-a-cookie-346519310e81

[3] [8] Security Best Practices for Express in Production
https://expressjs.com/en/advanced/best-practice-security.html

[4] javascript - Make Axios send cookies in its requests automatically - Stack Overflow
https://stackoverflow.com/questions/43002444/make-axios-send-cookies-in-its-requests-automatically

[5] [6] reactjs - How to refresh JWT tokens in React.js Application? - Stack Overflow
https://stackoverflow.com/questions/54531731/how-to-refresh-jwt-tokens-in-react-js-application

[7] Authentication store with zustand | Doichev Kostia
https://doichevkostia.dev/blog/authentication-store-with-zustand/