CSP Checklist – React & Next.js

Content Security Policy für moderne React-Anwendungen und Next.js Deployments

tenbyte

2025 / 2026

Next.js | React | Remix | Vite / SPA

Critical – vor Go-Live | Important | Optional | Next.js spezifisch | React / SPA

Wichtig: Immer gegen den **Production-Build** testen – Dev-Builds enthalten eval() und deutlich mehr Inline-Scripts. Report-Only Modus zuerst aktivieren, bevor blockiert wird.

01 Setup & Grundkonfiguration — Alle Frameworks

- [ ] **Report-Only Modus als ersten Schritt aktivieren** — Critical
  `Content-Security-Policy-Report-Only` setzen und mehrere Tage Violations sammeln, bevor man auf echte CSP wechselt.
- [ ] **Alle externen Ressourcen inventarisieren** — Critical
  Fonts, Analytics, CDNs, Error-Tracking (Sentry), Feature-Flags (LaunchDarkly), Chat-Widgets etc. vollständig auflisten.
- [ ] **default-src 'self' als Basis** — Critical
  Strenge Basis setzen, dann gezielt öffnen. Niemals mit `default-src *` starten.
- [ ] **object-src 'none' und base-uri 'self'** — Critical
  In React-Apps gibt es keinen Grund für `<object>` oder `<embed>`. `base-uri` verhindert Link-Hijacking via injiziertem `<base>`-Tag.
- [ ] **frame-ancestors 'none' oder eigene Domain** — Critical
  Clickjacking-Schutz. Nur öffnen wenn die App bewusst als iframe eingebettet wird.
- [ ] **Production-Build für alle CSP-Tests verwenden** — Critical
  Dev-Server und Production haben fundamental unterschiedliche Script-Anforderungen. Niemals Dev-Build als Referenz nehmen.

02 Nonce-Implementierung in Next.js (App Router) — Next.js

- [ ] **Nonce in middleware.ts generieren** — Critical, Next.js
  `crypto.randomUUID()` oder `crypto.getRandomValues()` bei jedem Request aufrufen. Nonce als `x-nonce` Response-Header und im CSP-Header setzen.
- [ ] **Nonce-Länge mind. 128 Bit (≥ 16 Bytes / ~22 Base64-Zeichen)** — Critical, Next.js
  Kürzere Nonces sind durch statistische Analyse oder Logs erratbar. `randomUUID()` liefert 122 Bit – ausreichend.
- [ ] **Nonce in layout.tsx via headers() lesen** — Critical, Next.js
  `headers()` aus `next/headers` in der Server Component aufrufen, Nonce an alle `<Script nonce={nonce}>` Komponenten übergeben.
- [ ] **Next.js Hydration-Scripts über Nonce freigeben** — Critical, Next.js
  Next.js injiziert Inline-Scripts für die Hydration. Ohne Nonce oder `'unsafe-inline'` schlägt die Hydration fehl. Nonce ist die sichere Lösung.
- [ ] **'strict-dynamic' in script-src ergänzen** — Important, Next.js
  Erlaubt Scripts die von einem nonce-verifizierten Script geladen werden (Chunks, Lazy Imports). Verhindert, dass jeder Chunk einzeln gewhitelistet werden muss.
- [ ] **Seiten mit Nonce vom Cache ausschließen** — Critical, Next.js
  Gecachte Responses würden denselben Nonce an alle Besucher ausliefern – womit die gesamte Sicherheit des Nonce entfällt.
- [ ] **Vercel Edge Runtime: crypto.getRandomValues() verwenden** — Important, Next.js
  Edge Runtime unterstützt nicht alle Node.js crypto APIs. `crypto.getRandomValues(new Uint8Array(16))` ist die zuverlässige Alternative.

03 Nonce-Implementierung – Pages Router & SSR — Next.js

- [ ] **Nonce in _document.tsx über getInitialProps generieren** — Critical, Next.js
  Im Pages Router in `_document.tsx` via `getInitialProps` pro Request einen Nonce erzeugen und in den Context übergeben.
- [ ] **CSP-Header in _document über res.setHeader setzen** — Critical, Next.js
  In `getInitialProps` auf das `ctx.res`-Objekt zugreifen und den Header inkl. Nonce dort setzen.
- [ ] **next/script mit nonce-Prop verwenden** — Important, Next.js
  `<Script nonce={nonce} strategy="beforeInteractive">` statt roher `<script>`-Tags – Next.js kümmert sich dann um die korrekte Platzierung.

04 Statische CSP via next.config.js — Next.js

- [ ] **headers()-Callback für Routen ohne Inline-Scripts** — Important, Next.js
  Für statische Seiten ohne dynamischen Inhalt: CSP über `headers()` in `next.config.js` setzen. Einfacher, aber kein Nonce möglich.
- [ ] **Separate Header-Regeln pro Route-Pattern** — Important, Next.js
  API-Routen (`/api/*`) brauchen andere CSP als Frontend-Routen. Mit `source`-Patterns granular steuern.
- [ ] **CDN-Domain für _next/static freigeben** — Important, Next.js
  Bei Custom-CDN oder Cloudflare: CDN-Domain in `script-src`, `style-src` und `img-src` eintragen.
- [ ] **X-Content-Type-Options und andere Security-Header ergänzen** — Important, Next.js
  `nosniff`, `Referrer-Policy`, `Permissions-Policy` direkt daneben setzen – gleicher Ort, gleiche Verantwortung.

05 React SPA / Vite / Create React App — React / Vite

- [ ] **CSP über den Webserver setzen (Nginx / Apache)** — Critical, React / SPA
  Statische SPAs haben kein Server-Backend. CSP-Header müssen am Webserver oder Hosting-Provider (Vercel, Netlify, Cloudflare Pages) gesetzt werden.
- [ ] **Netlify: _headers-Datei im public-Ordner** — Important, React / SPA
  `public/_headers` mit `/*` als Route und dem CSP-Header eintragen. Wird bei jedem Deploy automatisch angewendet.
- [ ] **Vercel: vercel.json headers-Block** — Important, React / SPA
  `vercel.json` mit `"headers"`-Array – CSP als Response-Header für alle Routen definieren.
- [ ] **Cloudflare Pages: _headers-Datei** — Important, React / SPA
  Identisch zu Netlify: `_headers`-Datei im Output-Verzeichnis. Alternativ über Cloudflare Transform Rules im Dashboard.
- [ ] **Vite Plugin für CSP-Hashes verwenden** — Important, React / SPA
  `vite-plugin-csp` oder `vite-plugin-sri` generieren automatisch SHA-256-Hashes für alle Inline-Scripts und Styles beim Build.
- [ ] **eval() in React DevTools Production checken** — Important, React / SPA
  React selbst braucht kein `eval()` in Production. Wenn 'unsafe-eval' nötig scheint, liegt es an einem Dependency – identifizieren und austauschen.

06 Script- und Ressourcen-Management — Alle Frameworks

- [ ] **next/script Strategy bewusst wählen** — Important, Next.js
  `beforeInteractive` blockiert Render, `afterInteractive` lädt nach Hydration, `lazyOnload` nach Load-Event, `worker` in Web Worker. Jede Strategy hat andere CSP-Implikationen.
- [ ] **Subresource Integrity (SRI) für CDN-Scripts** — Important
  `integrity="sha256-..."` bei externen Scripts hinzufügen. Browser führt Script nicht aus wenn Hash nicht stimmt – schützt vor CDN-Kompromittierung.
- [ ] **Google Tag Manager CSP-Anforderungen klären** — Important
  GTM braucht `'unsafe-inline'` oder Nonce. Alternativ: Server-Side GTM (sGTM) einsetzen – saubere CSP, alle Daten gehen über eigenen Server.
- [ ] **Sentry DSN in connect-src freigeben** — Important
  Sentry schickt Errors via fetch() an `*.sentry.io`. Ohne Eintrag in `connect-src` werden alle Error-Reports stillschweigend blockiert.
- [ ] **Matomo / Plausible in connect-src und script-src** — Important
  Self-hosted Matomo-Domain in `script-src` und `connect-src` freigeben. Bei Matomo Cloud: `*.matomo.cloud`.
- [ ] **WebSocket-Verbindungen in connect-src** — Important
  `wss://`-Domains für WebSockets explizit in `connect-src` eintragen. Next.js Dev-Server (`ws://localhost`) nur in Development freigeben.
- [ ] **Stripe.js / Payment-Iframes in frame-src** — Important
  `js.stripe.com` in `script-src`, `*.stripe.com` in `frame-src` und `connect-src`. PayPal, Klarna analog.
- [ ] **Google Fonts vollständig freigeben** — Optional
  `fonts.googleapis.com` in `style-src` UND `fonts.gstatic.com` in `font-src`. Oder: Fonts selbst hosten und Problem vermeiden.

07 Drittdienste & Embeds — Alle Frameworks

- [ ] **YouTube / Vimeo Embeds in frame-src** — Important
  `www.youtube-nocookie.com` (datenschutzfreundliche Variante) oder `www.youtube.com` in `frame-src`. Vimeo: `player.vimeo.com`.
- [ ] **Google Maps in frame-src und script-src** — Important
  `maps.googleapis.com` und `maps.gstatic.com` in `script-src` und `img-src`. Bei iframe-Embed: `www.google.com` in `frame-src`.
- [ ] **Hotjar / Clarity in script-src und connect-src** — Important
  `*.hotjar.com` in `script-src`, `connect-src` und `img-src`. Hotjar nutzt auch WebSockets – `wss://*.hotjar.com` in `connect-src`.
- [ ] **Intercom / Crisp / Live-Chat Widgets** — Important
  Chat-Widgets laden Scripts, Frames und API-Verbindungen. Anbieter-Dokumentation auf CSP-Anforderungen prüfen – oft 3–5 Domain-Einträge nötig.
- [ ] **Auth0 / Clerk / Supabase Auth in connect-src** — Important
  Auth-Dienste kommunizieren via fetch(). Ihre API-Domains müssen in `connect-src` stehen. Bei Redirect-basierten Flows auch `form-action` prüfen.
- [ ] **Supabase Storage / S3 / Cloudinary in img-src** — Optional
  Externe Image-Hosts in `img-src` freigeben. Bei Next.js Image-Optimization: `remotePatterns` in `next.config.js` und CSP synchron halten.

08 Testing & Validation

Alle Frameworks

- [ ] **CSP Evaluator (csp-evaluator.withgoogle.com)** — Critical
  Analysiert die Policy auf bekannte Bypass-Muster und Konfigurationsfehler. Vor Go-Live Pflicht.
- [ ] **Alle kritischen User Flows durchklicken** — Critical
  Login, Checkout, Formulare, Modals, Lazy-loaded Sections – nicht nur die Homepage. CSP-Violations treten oft nur auf bestimmten Routen auf.
- [ ] **Browser-Console nach Deployment prüfen** — Critical
  Keine CSP-Violations in der Console = grünes Licht. Auch in Firefox testen – unterschiedliche CSP-Implementierungen können verschiedene Errors zeigen.
- [ ] **securityheaders.com Scan durchführen** — Important
  Bewertet CSP und alle anderen Security-Header. Ziel: mindestens A-Rating. CSP alleine reicht nicht – weitere Header ergänzen.
- [ ] **report-uri Endpunkt für laufendes Monitoring** — Important
  Auch nach Go-Live Violations sammeln – neue Deployments oder Drittanbieter-Updates können neue Violations einführen.
- [ ] **Playwright/Cypress: Console-Errors in CI prüfen** — Optional
  E2E-Tests können auf CSP-Violations prüfen indem sie auf `console.error` Events mit "Content Security Policy" im Text hören.

09 Wartung & Lifecycle

Alle Frameworks

- [ ] **CSP-Policy als Konstante in der Codebase versionieren** — Important
  Policy nicht nur in Server-Configs vergraben. Als exportierte Konstante in `lib/csp.ts` oder ähnlich – dann in Middleware und Tests nutzbar.
- [ ] **Bei jedem neuen npm-Package CSP überprüfen** — Critical
  Neue Deps können Inline-Scripts, eval() oder externe Domains einführen. Package-README und Netzwerk-Tab nach Installation prüfen.
- [ ] **Bei Next.js Major-Upgrades CSP testen** — Critical
  Next.js ändert gelegentlich interne Script-Injection (z. B. Hydration, Turbopack). Nach Major-Upgrades immer neu testen.
- [ ] **Drittdienste auf Domain-Änderungen überwachen** — Important
  Analytics, Payment, Auth – Anbieter ändern gelegentlich ihre Domains. Violations-Reports frühzeitig aufzeigen.
- [ ] **Changelog für Policy-Änderungen führen** — Optional
  Warum wurde Domain X freigegeben? Wer hat es genehmigt? Wichtig für Security-Audits und Compliance.