Chapitre C — Performance & Accessibilité

5 skills regroupés sous ce thème.

- perf-cwv
- ux-flows
- a11y-wcag22-aa
- nielsen-audit
- lighthouse-ci-budgets

perf-cwv

perf-cwv

> *Catalogue généré le 2026-05-11*

En une phrase

Audit complet des Core Web Vitals (LCP, INP, CLS) d'une page web avec diagnostic, plan correctif priorisé et budgets de performance recommandés.

Quand l'utiliser

- Avant la mise en ligne d'une landing marketing ou d'une page SEO critique
- Quand une page semble lente, lourde ou saccadée
- Quand le score Lighthouse plafonne et qu'on ne sait plus quoi optimiser
- Pour fixer des budgets perf clairs (JS, CSS, images, fonts) sur un projet
- Avant un sprint de polish performance pour prioriser les actions à fort impact

Comment l'invoquer

- **Slash command** : `/perf-cwv`
- **Auto-invocation** : Oui — quand une page est lente, lourde, ou avant livraison marketing/SEO

Description détaillée

Ce skill suit une procédure structurée en quatre temps. D'abord il mesure (via Lighthouse ou des données réelles si elles existent), puis il diagnostique chacune des trois métriques Core Web Vitals

: le LCP (élément principal qui s'affiche, image hero, fonts, cache, SSR), l'INP (réactivité aux clics, hydration React/Vue, long tasks, taille de bundle) et le CLS (sauts visuels causés par des images sans dimensions, des fonts qui chargent tard, des injections dynamiques).

Ensuite il produit un plan d'actions priorisé en P0/P1/P2, avec les 5 chantiers à plus fort impact, et propose des budgets concrets pour ne pas laisser le projet regrasser après livraison. Il donne aussi des snippets prêts à coller (lazy loading, preload des ressources critiques, `aspect-ratio` pour les médias, code splitting).

Pour Thymon : c'est l'outil à dégainer quand un site fonctionne bien mais "rame", ou quand on veut s'assurer qu'une page marketing ne va pas couler au moment du lancement. Il est aussi utile en post-mortem si Lighthouse signale une régression après un déploiement.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `perf-cwv`
- **Fichier** : `/home/thymon/.claude/skills/perf-cwv/SKILL.md`

ux-flows

ux-flows

> *Catalogue généré le 2026-05-11*

En une phrase

Passe d'une idée floue à un plan d'interface complet : architecture d'information, écrans, parcours utilisateurs clés, états UX (loading/error/empty) et métriques de succès.

Quand l'utiliser

- Au démarrage d'un SaaS ou d'un dashboard pour structurer ce qu'on va construire
- Pour préparer un checkout, un onboarding ou un flow d'inscription
- Avant une refonte produit pour cartographier l'existant et les changements
- Sur une landing à fort enjeu de conversion qui doit guider vers une action
- Quand tu as un brief client mais zéro idée du nombre d'écrans à prévoir

Comment l'invoquer

- **Slash command** : `/ux-flows`
- **Auto-invocation** : Oui — pour structurer parcours UX, IA, écrans/sections

Description détaillée

Ce skill évite le piège classique du "je code direct" : foncer dans Figma ou dans React sans avoir clarifié ce que l'utilisateur vient chercher. Il commence par résumer la cible (qui), le contexte (où), l'objectif business (pourquoi) et les frictions actuelles s'il y en a.

Ensuite il identifie 3 à 5 Top Tasks — les choses concrètes pour lesquelles les utilisateurs viennent (par exemple "réserver une session", "comparer deux produits", "récupérer ma facture"). Puis il produit 2 à 3 flows clés sous forme d'étapes numérotées, en couvrant un happy path et au moins un edge case (paiement refusé, compte déjà existant, etc.). Il définit l'architecture d'information (navigation globale et locale, liste des écrans, blocs de contenu) et surtout n'oublie pas les états UX : loading, empty, error, success — qui sont 80 % du polish d'une UI mais que personne ne planifie au début.

Tu repars avec un document structuré qui couvre audience, top tasks, IA, flows numérotés, états et messages, plus une liste de risques UX avec leurs solutions. Pour Thymon : c'est le skill à utiliser dès qu'un projet dépasse "une seule page" — il transforme une discussion en plan actionnable avant de toucher au code.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `ux-flows`
- **Fichier** : `/home/thymon/.claude/skills/ux-flows/SKILL.md`

a11y-wcag22-aa

a11y-wcag22-aa

> *Catalogue généré le 2026-05-11*

En une phrase

Audit accessibilité complet contre le standard WCAG 2.2 niveau AA, avec checklist clavier, focus, contraste, formulaires, gestes et motion — à passer avant toute livraison production.

Quand l'utiliser

- Avant la mise en prod d'une app destinée à un public large
- Sur un site public où l'accessibilité a une obligation légale (administration, e-commerce, services)
- Quand tu veux vérifier qu'une UI nouvellement codée est navigable au clavier
- Pour préparer une remise client professionnelle où l'a11y est un livrable
- Après un sprint design pour valider que rien n'a régressé côté accessibilité

Comment l'invoquer

- **Slash command** : `/a11y-wcag22-aa`
- **Auto-invocation** : Oui — avant livraison prod

Description détaillée

Le skill déroule une checklist condensée qui couvre les 6 axes critiques de WCAG 2.2 AA. Premier axe : la navigation clavier — toutes les actions doivent être accessibles via

Tab/Shift+Tab/Enter/Space/Escape, sans piège clavier et avec un ordre logique. Deuxième axe : le focus, qui doit être visible, stylé et jamais masqué par un overlay ou un header sticky (c'est le nouveau critère "focus non obscurci" de la 2.2).

Suivent les tailles de cibles (les gros boutons pour pointeurs imprécis), les gestes (si tu utilises du drag-and-drop, prévois une alternative sans drag — autre nouveauté 2.2), le contraste de texte y compris dans les états hover/focus/disabled, les formulaires (labels liés et pas juste des placeholders, erreurs associées aux champs avec un message actionnable), et enfin le respect de `prefers-reduced-motion`.

Tu repars avec une liste hiérarchisée bloquants / majeurs / mineurs, des recommandations concrètes au niveau code (HTML, ARIA, CSS), et 5 à 10 tests manuels à exécuter avant de signer la livraison. Pour Thymon : c'est le filet de sécurité qui évite que ton site soit inutilisable pour 15 % de tes visiteurs, et le passage obligé sur tout projet client.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `a11y-wcag22-aa`
- **Fichier** : `/home/thymon/.claude/skills/a11y-wcag22-aa/SKILL.md`

nielsen-audit

nielsen-audit

> *Catalogue généré le 2026-05-11*

En une phrase

Audite une UI existante avec les 10 heuristiques UX de Jakob Nielsen (référence universelle depuis 1994) et produit un plan correctif priorisé en quick wins + chantiers structurels.

Quand l'utiliser

- Avant une refonte pour identifier ce qui ne va vraiment pas dans l'existant
- Quand la conversion d'une page est faible et qu'on ne sait pas pourquoi
- Après des retours utilisateurs négatifs pour formaliser les problèmes UX
- Pour préparer un dossier de remise client avec des critiques argumentées
- Sur ton propre projet, pour passer d'un "ça me semble bof" à des actions concrètes

Comment l'invoquer

- **Slash command** : `/nielsen-audit`
- **Auto-invocation** : Oui — pour refonte, conversion faible, retours utilisateurs négatifs

Description détaillée

Les 10 heuristiques de Nielsen sont la check-list UX la plus connue au monde : visibilité du statut système, correspondance entre l'app et le monde réel, contrôle et liberté de l'utilisateur, cohérence et standards, prévention des erreurs, reconnaissance plutôt que rappel, flexibilité et efficacité,

design minimaliste, aide à la récupération sur erreurs, aide et documentation.

Ce skill applique chaque heuristique aux pages que tu lui présentes. Pour chaque problème détecté, il produit une ligne structurée : heuristique concernée, constat précis, impact sur l'utilisateur, fix recommandé, effort de dev (S/M/L), priorité (P0/P1/P2). C'est cette double dimension Impact × Effort qui permet ensuite de prioriser intelligemment — un P0 facile passe avant un P0 difficile.

Il sort ensuite deux plans : un **"plan 7 jours"** de quick wins (corrections ≤ 1h ou ≤ 1j chacune, pour gagner vite en qualité ressentie) et un **"plan 30 jours"** de chantiers structurels (≥ 2j, pour les sujets qui demandent une vraie refonte d'écran ou de flow). Pour Thymon : c'est l'audit qu'on dégaine quand un client dit "ça marche mais c'est pas terrible" sans savoir formuler quoi. Tu reviens avec une liste claire de "voici les 6 vrais problèmes et voici par quoi commencer".

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `nielsen-audit`
- **Fichier** : `/home/thymon/.claude/skills/nielsen-audit/SKILL.md`

lighthouse-ci-budgets

lighthouse-ci-budgets

> *Catalogue généré le 2026-05-11*

En une phrase

Configure Lighthouse CI dans ton pipeline pour qu'il bloque automatiquement les régressions performance, accessibilité ou SEO via des assertions et budgets définis dans un fichier de config.

Quand l'utiliser

- Sur un projet avec CI/CD (GitHub Actions, GitLab CI) et plusieurs contributeurs
- Quand tu veux te protéger contre les régressions de perf après un merge
- Pour fixer des seuils minimum acceptables (score perf > 80, LCP < 2.5s, etc.)
- Sur un projet client qui a un engagement de qualité contractuel
- Pour automatiser ce qui se fait sinon à la main avant chaque déploiement

Comment l'invoquer

- **Slash command** : `/lighthouse-ci-budgets`
- **Auto-invocation** : Oui — quand un projet a CI/CD ou plusieurs contributeurs

Description détaillée

Lighthouse CI est la version automatisable de Lighthouse (l'outil intégré dans Chrome DevTools). Au lieu de cliquer dans le navigateur après chaque déploiement, tu le branches sur ta pipeline et il tourne tout seul, comparant les résultats à des seuils que tu as définis. Si une régression dépasse

le seuil, la pipeline échoue et le merge est bloqué.

Ce skill produit quatre choses concrètes : un fichier de config `lighthouserc.json` (ou `.lighthouserc.js` si tu préfères du JS) avec les assertions et budgets paramétrés, les scripts npm pour lancer Lighthouse CI en local et en CI (`lhci autorun`, `lhci collect`, `lhci assert`), un exemple GitHub Action prêt à coller dans `.github/workflows/lighthouse.yml`, et un set de **budgets initiaux réalistes** (taille JS, CSS, images, fonts) plus les seuils minimum (score perf, a11y, SEO, best practices).

Deux conseils d'usage importants : démarrer avec des seuils réalistes basés sur l'existant — si tu mets `perf > 95` alors que le site est à 72, tu vas être en CI rouge pour toujours — puis resserrer progressivement. Et activer le **multirun** (3 ou 5 exécutions par audit) pour réduire la variance naturelle de Lighthouse qui peut donner +/- 5 points entre deux runs.

Pour Thymon : utile sur les projets qui partent en production sérieuse et où tu veux dormir tranquille, moins pertinent sur un side project où tu fais déjà attention à la main.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `lighthouse-ci-budgets`
- **Fichier** : `/home/thymon/.claude/skills/lighthouse-ci-budgets/SKILL.md`