02 — Mes skills perso (standalone)

21 skills personnels standalone (motion, design, accessibilité, content).

    - tiptap-generator
    - documenter-webapp
    - llm-council
    - add-tcg

- □□ Index — 02 — Mes skills perso (standalone)

Chapitre A — Motion & Animation

10 skills regroupés sous ce thème.

motion-tokens

motion-tokens

> *Catalogue généré le 2026-05-11*

En une phrase

Génère un jeu de tokens d'animation (durées, easings, distances) versionnables en JSON + variables CSS, avec mapping prêt pour Motion (React) et GSAP.

Quand l'utiliser

- Au démarrage d'un projet quand on veut une grammaire d'animation cohérente
- Quand les animations existantes sont incohérentes (chaque dev choisit ses durées au pif)
- Pour unifier un design system qui couvre déjà couleurs/typo mais pas le mouvement
- Avant d'attaquer un site marketing avec plusieurs sections animées
- Pour préparer le terrain avant `motion-system` ou `motion-framer-patterns`

Comment l'invoquer

- **Slash command** : `/motion-tokens`
- **Auto-invocation** : □ Oui — dès qu'un design system ou theming motion est nécessaire

Description détaillée

Ce skill produit un fichier `tokens.motion.json` avec 5 à 7 durées de base (par exemple `fast: 120ms`, `base: 200ms`, `slow: 320ms`), 3 à 5 courbes d'easing nommées (`standard`, `accelerate`, `decelerate`) et un set de distances pour les translations.

Il génère ensuite trois sorties exploitables tout de suite : le JSON pour versionner dans Git, des variables CSS (`--motion-duration-base`, `--motion-ease-standard`) à poser dans `:root`, et un mapping côté JS pour Motion ou GSAP afin que tu puisses écrire `transition={{ duration: motion.duration.base, ease: motion.ease.standard }}` au lieu de coder des nombres en dur.

Il finit par poser des règles d'usage : quelle durée pour un hover, laquelle pour une ouverture de modal, quel easing pour quel type de mouvement. Pour Thymon : c'est la fondation à poser avant d'animer sérieusement quoi que ce soit. Une fois ces tokens en place, tu peux changer le ressenti motion de tout le site en modifiant un seul fichier.

Source

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

motion-three-r3f

motion-three-r3f

> *Catalogue généré le 2026-05-11*

En une phrase

Guide pratique pour intégrer de la 3D web avec Three.js et React Three Fiber sans plomber la performance, en maîtrisant la boucle de rendu et les règles de cleanup.

Quand l'utiliser

- Quand on veut un hero WebGL animé sur une landing
- Pour ajouter un visuel 3D interactif (carte, produit, scène) à une page
- Quand un projet Three.js existant rame ou chauffe le CPU
- Avant d'attaquer une scène complexe pour valider qu'on a vraiment besoin de 3D
- Pour structurer proprement une intégration React + Canvas avec gestion du frameloop

Comment l'invoquer

- **Slash command** : `/motion-three-r3f`
- **Auto-invocation** : □ Oui — quand un projet implique de la 3D dans le navigateur

Description détaillée

Ce skill commence par te poser la question qui sauve : as-tu vraiment besoin de Three.js, ou est-ce qu'une image SVG animée ferait l'affaire ? La 3D web est lourde et la majorité des intégrations gagneraient à être remplacées par autre chose. Si la 3D est justifiée, il passe à la mise en place.

Il configure d'abord la boucle de rendu en mode `frameloop="demand"` plutôt qu'en rendu permanent : la scène ne se redessine que quand quelque chose change, ce qui peut diviser la consommation CPU par dix sur un hero statique. Il centralise la logique d'animation dans un seul `useFrame` au lieu de multiplier les `requestAnimationFrame` parallèles. Et il insiste sur le cleanup et le `dispose()` des géométries/textures au démontage, sinon le navigateur garde tout en mémoire.

Tu repars avec une architecture recommandée pour ton Canvas, des snippets prêts à copier pour `frameloop` + `invalidate` + `useFrame`, et une checklist perf à dérouler avant la mise en prod. Pour Thymon : c'est le skill à utiliser dès que le mot "WebGL" ou "3D" apparaît dans un brief, pour éviter de partir dans un truc impossible à maintenir.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-three-r3f`
- **Fichier** : `/home/thymon/.claude/skills/motion-three-r3f/SKILL.md`

reduced-motion-a11y

reduced-motion-a11y

> *Catalogue généré le 2026-05-11*

En une phrase

Conçoit une stratégie `prefers-reduced-motion` complète pour que les utilisateurs sensibles au mouvement (vertige, migraines) puissent profiter du site sans casser le design.

Quand l'utiliser

- Sur tout site avec des animations un peu marquées (parallax, transitions de page, scroll-triggered)
- Avant la mise en prod pour cocher la case accessibilité WCAG
- Quand un utilisateur signale du motion sickness sur une page
- Pour structurer une variante "reduce" qui reste élégante (pas juste tout couper)
- Sur un projet où le motion est central à l'identité (sinon trop d'animations à gérer)

Comment l'invoquer

- **Slash command** : `/reduced-motion-a11y`
- **Auto-invocation** : □ Oui — pour mettre en place une stratégie reduced-motion

Description détaillée

Le réflexe naïf face à `prefers-reduced-motion: reduce`, c'est de tout désactiver d'un coup avec une media query qui assassine toutes les transitions. Ce skill propose mieux : une matrice où tu décides animation par animation ce qui est essentiel (un chargement qui doit rester visible), ornemental (un parallax qu'on peut couper) ou à remplacer par une variante apaisée (un slide qui devient un fade).

Il génère ensuite l'implémentation CSS via la media query plus un attribut data (`data-motion="reduce"`) pour pouvoir tester en local sans changer les préférences système, et passe en revue les pièges classiques : le focus qui saute pendant une transition de page, les modales qui s'ouvrent sans animation visible donc semblent surgir, les drawers qui ne préviennent plus l'utilisateur.

Tu repars avec la matrice normal vs reduce, des snippets CSS et utilitaires JS, et une checklist QA pour vérifier que le site reste utilisable en mode reduced-motion. Pour Thymon : à utiliser sur tout projet où il y a plus de trois animations significatives, surtout les sites marketing où les utilisateurs ne s'attendent pas à un crash de leur oreille interne.

Source

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

motion-visual-regression

motion-visual-regression

> *Catalogue généré le 2026-05-11*

En une phrase

Met en place des tests visuels Playwright (golden screenshots) pour détecter automatiquement quand une animation casse ou quand un composant change visuellement sans qu'on le veuille.

Quand l'utiliser

- Sur un projet où les régressions visuelles coûtent cher (site client, portfolio public)
- Quand une équipe travaille à plusieurs et qu'on veut un filet de sécurité en CI
- Après avoir polish un motion system pour figer le rendu attendu
- Quand on a déjà eu un bug "ça marche en local mais pas en prod" sur du visuel
- Sur les pages critiques d'un site (hero, checkout, formulaires)

Comment l'invoquer

- **Slash command** : `/motion-visual-regression`
- **Auto-invocation** : Oui — pour stabiliser le motion et détecter les régressions visuelles

Description détaillée

L'idée : Playwright prend une capture d'écran de référence (le "golden") quand tout va bien, puis à chaque commit il recompare la page actuelle à la référence. Si les pixels diffèrent au-delà d'un seuil, le test échoue et tu sais qu'un changement de CSS ou un refactor a impacté le visuel.

Ce skill installe Playwright, écrit deux premiers tests `toHaveScreenshot()` (un état idle + un état stable après animation), puis t'arme contre la "flakiness" — c'est-à-dire les faux positifs où le test échoue sans raison à cause d'une animation qui n'est pas finie au moment de la capture. Pour ça il propose de désactiver les animations pendant les tests, de masquer les zones dynamiques (timestamp, données live), et de fixer des seuils de tolérance pixel pertinents.

Tu repars avec des scripts npm prêts à lancer, deux tests qui fonctionnent dès le premier `npx playwright test`, et un set de recommandations pour éviter que ta suite de tests visuels devienne un cauchemar à maintenir. Pour Thymon : utile si tu déploies souvent et que tu veux dormir tranquille la nuit, moins pertinent sur un projet perso en solo où tu vois tout de suite si quelque chose casse.

Source

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

motion-perf-audit

motion-perf-audit

> *Catalogue généré le 2026-05-11*

En une phrase

Diagnostic ciblé d'une animation qui rame ou d'un scroll saccadé, avec identification de la cause (compositing, layout thrashing, will-change mal posé) et plan de correction priorisé.

Quand l'utiliser

- Quand une animation visiblement saccade ou "lag" sur certains appareils
- Quand le scroll d'une page sent le pâteux et qu'on ne sait pas pourquoi
- Si le CPU s'envole pendant une animation (ventilateur du laptop qui démarre)
- Après une mise en prod où une animation tournait bien en dev mais plus en réel
- Pour valider qu'un hero animé est viable avant de le shipper

Comment l'invoquer

- **Slash command** : `/motion-perf-audit`
- **Auto-invocation** : Oui — pour auditer une animation laggy

Description détaillée

La performance motion est un sujet contre-intuitif : ce n'est pas le nombre d'animations qui pose problème, c'est le type de propriétés animées. Animer `transform` et `opacity`, c'est gratuit (le navigateur délègue au GPU). Animer `top`, `left`, `width` ou `height`, c'est cher (chaque frame relance un calcul de layout).

Ce skill commence par classer le symptôme observé (jank ponctuel, scroll lourd, jank permanent, CPU haut), puis enquête sur les causes probables : propriétés animées qui forcent du layout, "thrashing" où on lit et écrit le DOM en boucle, plusieurs `requestAnimationFrame` qui se marchent dessus, observers (`IntersectionObserver`, `ResizeObserver`) qui se déclenchent en cascade, ou `will-change` posé partout "au cas où" (ce qui consomme de la mémoire GPU sans aider).

Il produit ensuite un tableau symptôme → cause → preuve → fix, un top 5 d'actions priorisées P0/P1/P2 avec snippets de code, et une Definition of Done mesurable (par exemple "scroll à 60fps sur Chrome desktop, 30fps minimum sur mobile mid-range"). Pour Thymon : c'est l'enquêteur à appeler quand un truc rame sans raison apparente, avant de jeter l'animation entière.

Source

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

motion-framer-patterns

motion-framer-patterns

> *Catalogue généré le 2026-05-11*

En une phrase

Snippets et patterns prêts à l'emploi pour la librairie Motion (anciennement Framer Motion) en React : animations d'entrée/sortie, layout animations, shared elements, gestures et reduced-motion.

Quand l'utiliser

- Pour ajouter une animation d'apparition propre à un composant React
- Quand tu veux qu'un élément "glisse" élégamment d'une grille vers un détail (layout animation)
- Pour faire un effet de partage entre deux pages (shared element transition)
- Quand un site React commence à avoir trop d'animations désordonnées
- Pour avoir une référence rapide des bonnes pratiques sans relire la doc de Motion

Comment l'invoquer

- **Slash command** : `/motion-framer-patterns`
- **Auto-invocation** : □ Oui — pour des patterns d'animation React avec Motion

Description détaillée

Motion (anciennement Framer Motion) est la librairie d'animation la plus utilisée en React. Elle est très puissante mais facile à mal utiliser, ce qui dégrade la performance et casse le ressenti.

Ce skill commence par t'aider à choisir le pattern adapté à ton besoin : `enter/exit` pour une apparition simple, `layout` pour qu'un élément s'adapte tout seul quand son conteneur change, `shared` pour qu'un élément "voyage" entre deux contextes (par exemple une carte qui s'ouvre en modal), `scroll` pour le scroll-triggered. Il applique ensuite les tokens du projet (durées, easings) pour rester cohérent avec le reste du motion system, et implémente la variante reduced-motion en parallèle.

Il finit par une checklist QA qui rappelle les pièges : ne pas avoir trop de `layout` animations simultanées (chacune coûte en perf), préférer `transform` à `top/left`, nettoyer les listeners au démontage. Tu repars avec trois snippets prêts à coller (un enter/exit, un layout, un shared), 5 do/don't maximum pour ne pas surcharger, et une note explicite sur le reduced-motion. Installation : `npm install motion`. Pour Thymon : c'est ta référence rapide pour ne pas perdre 30 minutes à chaque animation React.

Source

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

motion-gsap-scrolltrigger

motion-gsap-scrolltrigger

> *Catalogue généré le 2026-05-11*

En une phrase

Snippets et bonnes pratiques GSAP + ScrollTrigger pour animer au scroll : reveals progressifs, pinning d'éléments, scrub lié au scroll, timelines complexes, cleanup et refresh sur changements de layout.

Quand l'utiliser

- Pour un site marketing ou portfolio avec une narration scrollytellante
- Quand tu veux un élément qui reste collé pendant qu'un autre défile (pinning)
- Pour synchroniser une animation à la position exacte de scroll (scrub)
- Sur une landing où plusieurs sections doivent "se révéler" en cascade au scroll
- Pour fiabiliser un projet GSAP existant qui a des animations qui ne se déclenchent plus après un changement de layout

Comment l'invoquer

- **Slash command** : `/motion-gsap-scrolltrigger`
- **Auto-invocation** : □ Oui — pour des patterns GSAP + ScrollTrigger

Description détaillée

GSAP est la librairie d'animation la plus puissante du web, et son plugin ScrollTrigger est ce qu'on utilise dès qu'on veut faire du "scroll storytelling" (les sites Apple, Stripe Press, et la plupart des landings premium).

Ce skill couvre les trois grands patterns ScrollTrigger : `reveal` (un élément qui apparaît quand il entre dans la viewport), `scrub` (une animation dont le timeline est piloté directement par la position du scroll, ce qui crée des effets de "défilement contrôlé"), et `pin` (un élément qui reste en place pendant que le contenu défile derrière, technique signature des sites éditoriaux). Il insiste sur les pièges spécifiques à GSAP : ne pas créer des instances ScrollTrigger en double (la consommation explose vite), penser à appeler `ScrollTrigger.refresh()` quand le DOM change après le chargement (sinon les triggers sont calculés sur l'ancien layout), nettoyer toutes les animations au démontage du composant React/Vue pour éviter les fuites mémoire.

Il fournit aussi la variante `reduced-motion` : sur les patterns scrub/pin, on remplace généralement par un simple fade ou par un affichage instantané. Installation : `npm install gsap`. Pour Thymon : ce skill est ton kit de survie quand tu attaques un site avec du scroll narratif fort, comme un portfolio ou une landing premium.

Source

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

motion-view-transitions

motion-view-transitions

> *Catalogue généré le 2026-05-11*

En une phrase

Implémente les View Transitions natives du navigateur (SPA ou MPA) avec shared elements, en progressive enhancement — variante reduced-motion incluse et focus géré proprement.

Quand l'utiliser

- Pour ajouter des transitions natives entre pages d'un site multi-pages (MPA)
- Sur une SPA Vue/React quand on veut des transitions plus fluides que celles de la librairie
- Quand tu veux qu'un élément "voyage" entre deux pages (image qui s'agrandit en passant au détail)
- Pour donner un ressenti premium à un portfolio ou un site marketing
- Si tu veux les bénéfices visuels sans la dette technique d'une libraire externe

Comment l'invoquer

- **Slash command** : `/motion-view-transitions`
- **Auto-invocation** : □ Oui — pour implémenter des View Transitions avec shared elements

Description détaillée

L'API View Transitions est la nouveauté navigateur qui change la donne : avec quelques lignes de CSS et un appel à `document.startViewTransition()`, on obtient des transitions de pages fluides sans librairie, et même des animations d'éléments partagés entre deux états (le fameux "shared element transition" des apps mobiles).

Ce skill démarre par la stratégie de support et le fallback : tous les navigateurs ne la supportent pas encore (Safari traîne). Il met en place un progressive enhancement où les navigateurs récents profitent des transitions, et les autres ont une navigation classique qui marche normalement. Il configure ensuite la transition globale (fade ou slide subtil) puis les transitions d'éléments partagés via `view-transition-name` en CSS.

Deux points critiques que le skill couvre : la gestion du **focus**, qui doit suivre logiquement la transition pour ne pas perdre les utilisateurs au clavier, et la **variante reduced-motion** qui désactive l'animation tout en gardant le changement de page propre. Il termine par une checklist QA pour vérifier que la navigation back/forward du navigateur fonctionne aussi (souvent oublié).

Tu repars avec une stratégie progressive enhancement claire, un snippet minimal JS + CSS qui marche, la variante reduced-motion et la checklist de QA. Pour Thymon : c'est la version "motion" du skill, complémentaire de `view-transitions` qui est plus orientée perception utilisateur. Utilise celui-ci quand tu veux du shared element fluide.

Source

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

view-transitions

view-transitions

> *Catalogue généré le 2026-05-11*

En une phrase

Implémente l'API View Transitions du navigateur en progressive enhancement pour ajouter des transitions de pages fluides (fade, slide, shared element) sans casser l'accessibilité ni la performance.

Quand l'utiliser

- Sur un portfolio où la navigation entre projets doit avoir un ressenti premium
- Pour une landing marketing où on veut un effet "wow" subtil entre sections
- Sur une SPA Next/Nuxt/Astro qui veut des transitions sans librairie tierce
- Quand le client demande des transitions "comme les apps mobiles"
- Pour ajouter une couche de polish à un site déjà fonctionnel sans tout refaire

Comment l'invoquer

- **Slash command** : `/view-transitions`
- **Auto-invocation** : □ Oui — pour navigation premium, portfolios, marketing, apps

Description détaillée

C'est la version "perception utilisateur" du skill, là où `motion-view-transitions` est plus orientée motion designer. L'objectif ici est plus pragmatique : ajouter des transitions de pages **uniquement si elles apportent un bénéfice clair** (perception de fluidité, continuité visuelle entre deux états), pas juste pour faire joli.

Le skill commence par vérifier la compatibilité navigateur et définit une stratégie de fallback : sur les navigateurs qui ne supportent pas l'API (Safari traîne, Firefox arrive), la navigation reste classique. Il implémente ensuite des transitions simples — généralement un fade ou un slide subtil — et propose une variante "shared element" pour les cas où un élément doit visuellement persister entre deux pages.

Deux préoccupations transverses guident l'implémentation : **préserver le focus** (l'utilisateur clavier ne doit pas être perdu après une transition) et **respecter** `prefers-reduced-motion` (transition désactivée pour les utilisateurs sensibles au mouvement). Tu repars avec une stratégie support+fallback claire, un snippet minimal JS + CSS, les variantes fade/shared element, et les notes d'accessibilité associées.

Pour Thymon : c'est le skill rapide à dégainer quand tu veux ajouter une touche premium à un projet sans installer Motion ou GSAP. Plus léger, plus moderne, suffisant dans 80 % des cas.

Source

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

motion-system

motion-system

> *Catalogue généré le 2026-05-11*

En une phrase

Conçoit un motion system complet pour un projet : principes directeurs, tokens (durations/easings/distances), 6 patterns réutilisables (enter, hover, focus, modal, list, page) et règles reduced-motion intégrées.

Quand l'utiliser

- Sur un projet ambitieux où le motion fait partie de l'identité (portfolio, marketing premium, app produit)
- Quand `motion-tokens` ne suffit plus et qu'il faut formaliser les patterns
- Pour aligner une équipe sur "quelle animation pour quel usage" avant de commencer à coder
- Avant de générer la doc d'un design system pour avoir un chapitre motion solide
- Sur une refonte de site où le mouvement est un levier de différenciation

Comment l'invoquer

- **Slash command** : `/motion-system`
- **Auto-invocation** : □ Oui — pour produire une grammaire motion cohérente d'un projet

Description détaillée

Un motion system, c'est l'équivalent du design system mais pour le mouvement. Au lieu d'avoir 47 animations différentes choisies au feeling, on a un vocabulaire restreint et cohérent qui se réutilise partout — ce qui rend le site mémorable et facile à maintenir.

Ce skill produit le motion system en 4 temps. D'abord les **intentions** : pour quoi anime-t-on ? Hiérarchie (un élément attire l'œil), causalité (un clic ici déclenche un changement là), feedback (l'app me dit qu'elle a compris), émotion (un détail délicieux). Ensuite les **tokens** : 5-7 durations et 3-5 easings nommés sémantiquement, plus des distances standard. Puis 6 **patterns** prêts à l'emploi : enter (apparition à l'arrivée d'un élément), hover (réaction au survol), focus (signal clavier), modal (ouverture/fermeture), list (animations en série sur une liste), page (transition d'écran). Enfin les **règles reduced-motion**, déclinées pattern par pattern (pas juste un kill switch global).

À la sortie, tu as 6-10 lignes de principes directeurs, les tokens en JSON + CSS variables, les 6 patterns avec snippets fonctionnels, et une matrice normal vs reduce pour chacun. Conseil intégré : préférer `transform` et `opacity` (gratuits pour le GPU), éviter `width` / `height` / `top` / `left` qui forcent du layout. Pour Thymon : c'est le skill "boss" du motion. Une fois ce système posé, tous les autres skills motion (framer-patterns, gsap, view-transitions) viennent s'y appuyer naturellement.

Source

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

Chapitre B — Design & Direction artistique

2 skills regroupés sous ce thème.

design-tokens

design-tokens

> *Catalogue généré le 2026-05-11*

En une phrase

Produit un système de design tokens versionnable (couleurs, typo, espacements, radius, ombres) avec dark mode intégré, livré en JSON + variables CSS + mapping Tailwind prêts à brancher.

Quand l'utiliser

- Au démarrage de tout projet qui aura plus de quelques composants
- Quand un projet existant a des couleurs et tailles éparpillées partout dans le code
- Pour préparer un design system avant d'attaquer la création des composants
- Quand tu veux ajouter un mode sombre proprement sans tout refaire
- Si tu utilises Tailwind et veux que les classes (`bg-primary`, `text-muted`) pointent vers des variables CSS modifiables

Comment l'invoquer

- **Slash command** : `/design-tokens`
- **Auto-invocation** : □ Oui — dès qu'un design system ou theming est nécessaire

Description détaillée

Un design token, c'est un nom pour une valeur. Au lieu d'écrire `#3B82F6` à 47 endroits dans ton CSS, tu écris `var(--color-primary)` partout, et la couleur est définie une seule fois. Si tu décides de la changer, tu modifies une ligne et tout le site suit.

Ce skill suit deux règles fondamentales. D'abord il sépare les tokens **bruts** (les valeurs : `blue-500: #3B82F6`) des tokens **sémantiques** (les rôles : `primary: var(--blue-500)`). C'est cette séparation qui permet de changer la palette en gardant la sémantique, ou inversement. Ensuite il prévoit toujours le dark mode dès le départ — pas des couleurs inversées au pif, mais les mêmes rôles avec des valeurs adaptées.

Il génère ensuite trois sorties cohérentes entre elles : un `tokens.json` à versionner dans Git, des CSS variables sur `:root` et `[data-theme="dark"]`, et un snippet de configuration Tailwind avec `colors: { primary: 'hsl(var(--color-primary))' }`. Les rôles couverts incluent les couleurs (bg/surface/elevated/text/muted/border/primary/accent/danger/success/warn), la typo (display vs body, sizes, leading), le spacing (base 4 ou 8), les radius (xs à 2xl), les shadows et optionnellement le motion. Pour Thymon : c'est la fondation d'un projet web sérieux, à poser dès qu'il y a une chance de passer en dark mode ou de devoir refondre les couleurs plus tard.

Source

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

art-direction

art-direction

> *Catalogue généré le 2026-05-11*

En une phrase

Transforme un brief vague ("je veux un truc moderne et chaleureux") en direction artistique web complète et actionnable : palette, typographie, layout, motion, imagerie — plus une alternative contrastée pour comparer.

Quand l'utiliser

- Au tout démarrage d'un projet quand on n'a que des mots flous comme brief
- Pour proposer deux directions visuelles différentes à un client et le faire trancher
- Quand on veut sortir des sentiers battus AI-slop et donner une vraie identité au site
- Avant de plonger dans Figma ou dans le code, pour cadrer les décisions visuelles
- Pour une refonte où il faut articuler "pourquoi ce nouveau look et pas l'ancien"

Comment l'invoquer

- **Slash command** : `/art-direction`
- **Auto-invocation** : □ Oui — au démarrage d'une création ou refonte

Description détaillée

La direction artistique est l'étape qu'on saute toujours quand on est pressé, et qu'on regrette ensuite quand chaque écran fait un compromis différent. Ce skill la formalise en quatre temps.

D'abord il extrait du brief les éléments structurants : audience cible, promesse, ton de voix, contraintes techniques (stack, délais, contenus à présenter). Ensuite il propose **deux directions artistiques contrastées** — pas deux variantes proches, mais deux pistes vraiment différentes (par exemple "éditorial sobre serif" vs "brutal coloré sans-serif déformée"). Chaque direction comporte 5 mood words, une phrase signature, 3 règles non négociables, et un pack prêt-dev complet : palette par rôles, typo pairing display+body avec tailles `clamp()` et line-heights, échelle d'espacements, radius, ombres, motifs UI (bento, editorial, brutal, glass, organique), principes motion.

À la sortie, tu as deux fiches DA structurées identiquement (pour comparer terme à terme), plus un "tokens quick draft" en JSON minimal et CSS variables prêtes pour les modes light + dark. Pour Thymon : c'est le skill à dégainer au tout début d'un projet, avant même `design-tokens` ou `frontend-design`. Il transforme une discussion floue en deux pistes concrètes qu'on peut juger avec ses yeux.

Source

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

Chapitre C — Performance & Accessibilité

5 skills regroupés sous ce thème.

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`

Chapitre D — Content & Outils dev

4 skills regroupés sous ce thème.

Chapitre D — Content & Outils dev

tiptap-generator

tiptap-generator

> *Catalogue généré le 2026-05-11*

En une phrase

Génère et valide automatiquement le JSON TipTap pour les articles du site éducatif d'espagnol juanmaycompania, en respectant la spec exacte de l'éditeur et en intégrant images, audio et PDF.

Quand l'utiliser

- Quand tu veux créer un nouvel article ou une nouvelle leçon sur juanmaycompania
- Pour transformer un texte brut en contenu structuré avec encarts, onglets, flip cards, timeline
- Quand tu as des images, audios ou PDFs à intégrer dans une leçon
- Pour mettre à jour le contenu d'un article existant en gardant le format JSON propre
- Avant un import en base pour t'assurer que le JSON sera accepté par l'éditeur

Comment l'invoquer

- **Slash command** : `/tiptap-generator`
- **Phrases déclencheurs (texte)** : "génère un article TipTap", "crée du contenu pour juanmaycompania", "JSON pour l'éditeur"
- **Auto-invocation** : □ Oui — quand on demande de générer un article, du contenu TipTap, ou du JSON pour l'éditeur

Description détaillée

TipTap est l'éditeur de contenu utilisé par juanmaycompania, et son format de stockage est un JSON arborescent assez strict. Une virgule mal placée et l'article entier ne s'affiche plus. Ce skill évite ce risque en suivant un workflow obligatoire.

Première étape : il lit la spec complète (`~/.claude/skills/tiptap-generator/spec/tiptap-mcp-spec.md`) qui décrit tous les nœuds autorisés (`heading`, `paragraph`, `tabs`, `flipCard`, `timeline`, `customImage`, `audio`, `pdf`, etc.) et leurs attributs. Deuxième étape : il génère le JSON conformément à cette spec. Troisième étape, cruciale : il valide le résultat en l'envoyant à `validate.js`, qui détecte les erreurs de schéma. S'il y a des erreurs, il corrige et re-valide en boucle jusqu'à ce que ça passe.

Il connaît les règles absolues : racine toujours `{ "type": "doc", "content": [...] }`, pas de HTML inline dans les textes (on utilise les marks), `heading.level` limité à 2/3/4, et les attributs JSON stringifiés des composants composites (`tabs.tabs`, `flipCard.cards`, `timeline.items`) qui sont des strings JSON échappées. Il intègre automatiquement les médias quand il voit les tags `[IMAGE DISPONIBLE — url: X]`, `[AUDIO DISPONIBLE — url: X]` ou `[PDF DISPONIBLE — url: X]`. Pour Thymon : c'est ton générateur de contenu attitré pour le site espagnol, qui te permet de partir d'un texte brut et d'avoir un JSON valide sans toucher au code.

Source

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

documenter-webapp

documenter-webapp

> *Catalogue généré le 2026-05-11*

En une phrase

Skill maison qui scanne le code source d'une webapp, génère une documentation complète (mode d'emploi utilisateur + doc technique), et la pousse automatiquement sur l'instance BookStack auto-hébergée de Thymon.

Quand l'utiliser

- Pour documenter une webapp entière depuis zéro (boardgame-referee, basketball-overlay, juanmaycompania, wishlist manager)
- Pour mettre à jour la doc d'un projet existant après une grosse évolution
- Quand un collègue ou un client doit accéder à un mode d'emploi sans voir la doc technique
- Pour produire un audit documentaire (qu'est-ce qui manque, qu'est-ce qui est obsolète)
- Quand tu veux centraliser la doc de tous tes projets au même endroit (bookstack.thymon.fr)

Comment l'invoquer

- **Slash command** : `/documenter-webapp`
- **Phrases déclencheurs (texte)** : "documente ce projet", "fais la doc", "mets à jour la doc de X", "génère le mode d'emploi", "audit documentaire"

- **Auto-invocation** : □ Oui — dès qu'on parle de documenter une webapp ou de pousser de la doc sur BookStack

Description détaillée

C'est ton propre outil de documentation, et il est déjà testé sur tes projets. Il applique systématiquement la même structure dans BookStack : une **Étagère** au nom du projet, qui contient **deux Livres** — un "Mode d'emploi" pour les utilisateurs finaux (collègues, communauté JeVeux, clients d'Il Pinocchio) et une "Documentation technique" pour toi et les futurs devs/ops. Cette séparation existe parce que BookStack gère les permissions au niveau Livre : tu peux ouvrir le mode d'emploi à des non-techs sans exposer la doc technique.

Le workflow est rigoureux. Phase scoping : il te pose 4 questions (quel projet, quel périmètre, quel public du mode d'emploi, captures dispos ?). Phase scan : il lit `README`, `package.json`, `Dockerfile`, structure de dossiers, point d'entrée backend, schéma BDD, routeur frontend, `.env.example`, middlewares d'auth — dans cet ordre précis. Phase génération : il écrit le contenu en Markdown CommonMark dans un dossier temporaire qui reflète la hiérarchie BookStack cible, chaque page datée et écrite pour son public (jargon-free pour l'utilisateur, exécutable pour la technique). Phase preview : il te montre l'arborescence cible et te demande validation **avant** de pousser. Phase push : via `bookstack_client.py`, opérations idempotentes basées sur titre+parent — relancer ne duplique rien.

Il a des garde-fous : jamais de suppression sans confirmation nominative, jamais d'invention de contenu (préférer une section "À compléter" honnête), jamais de secrets copiés. Pour Thymon : c'est l'outil qui garde toute ta connaissance projet centralisée et à jour, sans que tu aies à réinventer la roue à chaque webapp.

Source

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

llm-council

llm-council

> *Catalogue généré le 2026-05-11*

En une phrase

Soumet une décision importante à un conseil de 5 IA avec des angles de pensée différents, qui débattent, se relisent anonymement, puis produisent un verdict synthétisé par un président — méthode inspirée du LLM Council d'Andrej Karpathy.

Quand l'utiliser

- Pour une décision business à fort enjeu (pivoter, niche, positionnement, pricing)
- Quand tu hésites entre 2 ou 3 options stratégiques avec de vrais tradeoffs
- Pour pressure-tester une idée avant de te lancer ("est-ce que je suis fou ?")
- Quand tu veux des perspectives qui clashent au lieu d'un avis lisse et consensuel
- Pour challenger une copy de landing page, un positionnement, une décision d'embauche

Comment l'invoquer

- **Slash command** : `/llm-council`
- **Phrases déclencheurs (texte)** : "council this", "run the council", "war room this", "pressure-test this", "stress-test this", "debate this", "should I X or Y", "which option", "I'm torn between"
- **Auto-invocation** : Oui — quand une décision concrète avec des stakes réels est présentée

Description détaillée

Le principe : une seule IA, c'est une seule perspective. Tu peux avoir une bonne réponse ou une réponse moyenne sans moyen de le savoir. Le council corrige ça en convoquant **5 conseillers** spécialisés en angles de pensée qui se contredisent naturellement.

Les cinq conseillers : **The Contrarian** cherche le défaut fatal, ce qui va casser ; **The First Principles Thinker** ignore la question de surface et reformule le vrai problème ; **The Expansionist** cherche l'upside que tout le monde rate ; **The Outsider** n'a aucun contexte et réagit avec un œil frais ; **The Executor** ne pense qu'à "qu'est-ce qu'on fait lundi matin ?". Ces cinq créent trois tensions saines : downside vs upside, repenser vs exécuter, expert vs débutant.

Le workflow se déroule en 4 étapes. Step 1 : le skill scanne ton workspace (CLAUDE.md, memory, transcripts précédents) pour donner du contexte aux conseillers, puis reformule ta question en prompt neutre. Step 2 : il lance les 5 conseillers en parallèle (sub-agents) qui répondent indépendamment, sans nuance ni hedge. Step 3 : peer review — les 5 réponses sont anonymisées (Réponse A à E) et chaque conseiller juge les autres : "qui est le plus fort ? Quel est le plus gros blind spot ? Qu'est-ce qu'ils ont tous raté ?". Step 4 : un chairman synthétise tout en 5 sections : Where the Council Agrees / Where it Clashes / Blind Spots Caught / The Recommendation (claire, pas "ça dépend") / The One Thing to Do First.

Tu repars avec deux artefacts : un rapport HTML visuel auto-ouvert (`council-report-<timestamp>.html`) et un transcript Markdown complet pour creuser. Pour Thymon : c'est l'outil à dégainer quand tu hésites vraiment sur un truc qui compte. Ne l'utilise pas pour des questions à réponse unique ("c'est quoi la capitale de la France ?") — il est fait pour les vraies décisions avec incertitude.

Source

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

add-tcg

add-tcg

> *Catalogue généré le 2026-05-11*

En une phrase

Checklist 6-axes pour intégrer un nouveau Trading Card Game (Lorcana, Pokémon, One Piece, Yu-Gi-Oh, Marvel Snap…) dans l'app boardgame-referee, avec les chemins de fichiers exacts, les patterns à dupliquer et les vérifications end-to-end.

Quand l'utiliser

- Quand tu veux ajouter un nouveau TCG complet au projet boardgame-referee
- Pour intégrer un jeu de plateau plus simple (Ark Nova, Terraforming Mars) — version réduite à 2 axes
- Pour valider qu'une intégration TCG est vraiment terminée et fonctionnelle
- Pour cadrer le niveau d'intégration souhaité (Minimum / Standard / Full premium)
- Quand l'utilisateur évoque juste un nom de TCG ("et si je voulais Lorcana ?") sans dire "ajouter"

Comment l'invoquer

- **Slash command** : `/add-tcg`
- **Phrases déclencheurs (texte)** : "ajouter un nouveau TCG", "intégrer Lorcana", "support Pokemon TCG", "nouveau jeu de cartes", "ajouter One Piece TCG", "Sorcery Contested Realm", "et si je voulais Lorcana ?"

- **Auto-invocation** : □ Oui — dès qu'un nom de TCG est mentionné dans le contexte de boardgame-referee

Description détaillée

Ce skill est spécifique au projet **boardgame-referee**. Il découpe l'intégration d'un nouveau TCG en 6 axes indépendants : **(1) Cartes** (source de données, ingestion, autocomplete `@`), **(2) Symboles & UI** (icônes inline, modal zoom), **(3) Decompose-query** (parsing des questions utilisateur), **(4) Deckbuilding** (génération de decklists complètes), **(5) Set matching** (filtrage par extension), **(6) Méta** (tier lists, tournois). L'axe 1 est la fondation ; les autres peuvent être appliqués selon le niveau d'intégration choisi.

Avant de coder, le skill **pose 7 questions obligatoires** : source des données cartes (API live, package npm, dataset GitHub, scraping), multilinguisme (FR/EN/bilingue), tailles de deck officielles par format, système de couleurs/factions, symboles inline à rendre, source méta disponible, et si le PDF de règles a déjà été ingéré via `/add-game`. Sans ces réponses, il refuse de commencer — les hard-codes (couleurs, types, deck sizes) en dépendent.

Trois niveaux d'intégration sont proposés : **Minimum viable** (axes 1+2, pour un jeu de plateau), **Standard** (axes 1+2+3, TCG sans deckbuilding), **Full premium** (les 6 axes, niveau MTG/Riftbound). Il fournit un tableau d'aiguillage avec durée typique de chaque axe et les fichiers de référence à lire (`references/axe-1-cards.md`, etc.). Après tous les axes, il déroule 7 vérifications end-to-end : autocomplete fonctionnel, symboles rendus, synergie classifiée, deckbuilding validé, méta synchronisée, set matching filtré, auto-wrap des noms de cartes. Il rappelle aussi la **règle d'or** : ne jamais mapper sémantiquement les stats d'un TCG à celles d'un autre (chaque jeu a ses propres champs `card_<jeu>_<stat>`), sinon le LLM hallucine des analogies fausses entre jeux.

Pour Thymon : c'est ton kit de travail dédié quand tu veux étendre boardgame-referee à un nouveau jeu de cartes sans rater une étape.

Source

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

□□ Index — 02 — Mes skills perso (standalone)

02 — Mes skills perso (standalone)

> Index des skills regroupés dans ce livre. Clique sur un skill pour ouvrir sa fiche.

21 skills documentés.

Chapitre A — Motion & Animation

Skill	En une phrase
motion-framer-patterns	Snippets et patterns prêts à l'emploi pour la librairie Motion (anciennement Framer Motion) en React : animations…
motion-gsap-scrolltrigger	Snippets et bonnes pratiques GSAP + ScrollTrigger pour animer au scroll : reveals progressifs, pinning d'éléments, scrub lié au…
motion-perf-audit	Diagnostic ciblé d'une animation qui rame ou d'un scroll saccadé, avec identification de la cause (compositing, layout thrashing,…
motion-system	Conçoit un motion system complet pour un projet : principes directeurs, tokens (durations/easings/distances), 6 patterns…
motion-three-r3f	Guide pratique pour intégrer de la 3D web avec Three.js et React Three Fiber sans plomber la performance, en maîtrisant la boucle…

Skill	En une phrase
motion-tokens	Génère un jeu de tokens d'animation (durées, easings, distances) versionnables en JSON + variables CSS, avec mapping prêt pour…
motion-view-transitions	Implémente les View Transitions natives du navigateur (SPA ou MPA) avec shared elements, en progressive enhancement — variante…
motion-visual-regression	Met en place des tests visuels Playwright (golden screenshots) pour détecter automatiquement quand une animation casse ou quand…
reduced-motion-a11y	Conçoit une stratégie `prefers-reduced-motion` complète pour que les utilisateurs sensibles au mouvement (vertige, migraines)…
view-transitions	Implémente l'API View Transitions du navigateur en progressive enhancement pour ajouter des transitions de pages fluides (fade,…

Chapitre B — Design & Direction artistique

Skill	En une phrase
art-direction	Transforme un brief vague ("je veux un truc moderne et chaleureux") en direction artistique web complète et actionnable :…
design-tokens	Produit un système de design tokens versionnable (couleurs, typo, espacements, radius, ombres) avec dark mode intégré, livré en…

Chapitre C — Performance & Accessibilité

Skill	En une phrase
a11y-wcag22-aa	Audit accessibilité complet contre le standard WCAG 2.2 niveau AA, avec checklist clavier, focus, contraste, formulaires, gestes…

Skill	En une phrase
lighthouse-ci-budgets	Configure Lighthouse CI dans ton pipeline pour qu'il bloque automatiquement les régressions performance, accessibilité ou SEO via…
nielsen-audit	Audite une UI existante avec les 10 heuristiques UX de Jakob Nielsen (référence universelle depuis 1994) et produit un plan…
perf-cwv	Audit complet des Core Web Vitals (LCP, INP, CLS) d'une page web avec diagnostic, plan correctif priorisé et budgets de…
ux-flows	Passe d'une idée floue à un plan d'interface complet : architecture d'information, écrans, parcours utilisateurs clés, états UX…

Chapitre D — Content & Outils dev

Skill	En une phrase
add-tcg	Checklist 6-axes pour intégrer un nouveau Trading Card Game (Lorcana, Pokémon, One Piece, Yu-Gi-Oh, Marvel Snap…) dans l'app…
documenter-webapp	Skill maison qui scanne le code source d'une webapp, génère une documentation complète (mode d'emploi utilisateur + doc…
llm-council	Soumet une décision importante à un conseil de 5 IA avec des angles de pensée différents, qui débattent, se relisent anonymement,…
tiptap-generator	Génère et valide automatiquement le JSON TipTap pour les articles du site éducatif d'espagnol juanmaycompania, en respectant la…