Chapitre D — Content & Outils dev

4 skills regroupés sous ce thème.

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

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`