Chapitre D — Planning & Stratégie

6 skills regroupés sous ce thème.

- plan-tune
- plan-ceo-review
- autoplan
- plan-eng-review
- plan-devex-review
- office-hours

plan-tune

plan-tune

> *Catalogue généré le 2026-05-11*

En une phrase

Règle la fréquence des questions que Claude te pose et inspecte ton profil de développeur (ce que tu as déclaré aimer vs ce que ton comportement révèle).

Quand l'utiliser

- Quand Claude te pose toujours la même question agaçante et que tu veux qu'il arrête
- Quand tu veux voir ton profil : tes préférences (rapide vs minutieux, scope large vs petit, etc.)
- Quand tu veux changer ta posture : « je veux être consulté à chaque décision » ou au contraire « délègue, fais au mieux »
- Quand tu veux voir l'historique des questions que Claude t'a posées dans ce projet
- Quand tu veux activer ou désactiver complètement le système de réglage de questions

Comment l'invoquer

- **Slash command** : `/plan-tune` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "tune questions" / "stop asking me that" / "too many questions" / "show my profile" / "show my vibe" / "developer profile" / "turn off question tuning"
- **Auto-invocation** : □ Oui — Claude propose ce skill quand il remarque que la même question revient ou que tu surcharges souvent ses recommandations.

Description détaillée

`/plan-tune` est un panneau de réglages conversationnel : tu parles en langage naturel, pas besoin de mémoriser des sous-commandes. Tu peux dire « montre-moi mon profil », « arrête de me demander ça », « passe-moi en mode délègue » et Claude comprend.

Le skill gère deux choses. D'un côté, un registre de toutes les questions que les autres skills posent (validation, choix de scope, routage), avec trois préférences possibles : ne jamais demander, demander à chaque fois, demander seulement pour les décisions irréversibles. De l'autre, un profil développeur en cinq dimensions : appétit de scope (petit vs « boil the ocean »), tolérance au risque, préférence de détail (réponses courtes vs explications), autonomie (consulter vs déléguer), soin de l'architecture.

Ton profil a deux pistes : celui que tu as déclaré toi-même et celui que ton comportement réel suggère. Quand il y a un écart, le skill te montre la différence (« tu te dis prudent mais tes choix sont rapides »). Cette version 1 est purement observationnelle : aucun autre skill ne modifie son comportement à partir du profil — c'est juste un miroir pour t'aider à mieux te connaître et désamorcer les questions répétitives.

Source

- **Plugin** : `gstack`
- **Nom interne** : `plan-tune`
- **Fichier** : `/home/thymon/.claude/skills/gstack/plan-tune/SKILL.md`

plan-ceo-review

plan-ceo-review

> *Catalogue généré le 2026-05-11*

En une phrase

Une relecture de plan en mode « PDG/fondateur » : Claude challenge ton ambition, repense le problème, et te pousse vers le produit dix étoiles.

Quand l'utiliser

- Quand tu as un plan ou un document de design et tu te demandes s'il est assez ambitieux
- Quand tu veux qu'on remette en cause les prémisses (« et si on faisait carrément l'inverse ? »)
- Quand tu hésites entre « tout faire d'un coup » et « livrer le minimum »
- Quand tu veux quelqu'un qui pense à 5-10 ans, pas seulement au prochain sprint
- Avant de coder, pour t'assurer que tu construis la bonne chose, pas juste « la bonne chose qu'on t'a demandée »

Comment l'invoquer

- **Slash command** : `/plan-ceo-review` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "think bigger" / "expand scope" / "strategy review" / "rethink this plan" / "is this ambitious enough"
- **Auto-invocation** : Oui — Claude propose ce skill quand tu questionnes l'ambition d'un plan ou que ton projet sent qu'il pourrait viser plus haut.

Description détaillée

`/plan-ceo-review` lit ton plan ou ton document de design et l'analyse comme le ferait un fondateur expérimenté (style YC, Stripe, Apple). Tu choisis d'abord l'une des quatre postures : EXPANSION DE SCOPE (rêver grand, ajouter ce qui rendrait le produit dix fois meilleur), EXPANSION SÉLECTIVE (tenir le scope mais piocher les meilleures idées), TENIR LE SCOPE (rigueur maximale sans bouger l'objectif), RÉDUCTION (couper jusqu'à l'essentiel).

Le skill applique seize « instincts cognitifs » de grands dirigeants : portes réversibles vs irréversibles (Bezos), focus comme soustraction (Jobs), inversion (« qu'est-ce qui nous ferait échouer ? »), paranoïa stratégique (Grove), volonté comme stratégie (Altman). Il fait un audit système de ton repo avant de commencer (historique git, TODOs, design docs existants), puis te pose des questions une par une via un format de décision structuré : titre, explication simple, recommandation, pour/contre, conséquence si tu te trompes.

Le résultat est un plan affiné avec un rapport « GSTACK REVIEW REPORT » ajouté à la fin. Aucun code n'est touché — la sortie c'est un meilleur plan, prêt à passer ensuite à `/plan-eng-review` pour la partie technique.

Source

- **Plugin** : `gstack`
- **Nom interne** : `plan-ceo-review`
- **Fichier** : `/home/thymon/.claude/skills/gstack/plan-ceo-review/SKILL.md`

autoplan

autoplan

> *Catalogue généré le 2026-05-11*

En une phrase

Lance d'un coup toute la chaîne de relectures (PDG, design, ingénierie, expérience développeur) en laissant Claude décider lui-même des choix intermédiaires.

Quand l'utiliser

- Quand tu as un plan brouillon et tu veux qu'il passe par toutes les relectures sans répondre à 15-30 questions intermédiaires
- Quand tu fais confiance à Claude pour les décisions de routine et tu veux juste arbitrer les vraies questions de goût à la fin
- Quand tu n'as pas le temps de faire `/plan-ceo-review` puis `/plan-eng-review` puis `/plan-devex-review` à la main
- Pour un plan déjà mûr où tu veux maximum de rigueur sans intervention constante
- Quand tu veux la « rolls » du planning : sortie = plan entièrement validé

Comment l'invoquer

- **Slash command** : `/autoplan` (à taper dans Claude Code)
- **Voice triggers** : « auto plan » · « automatic review »
- **Phrases déclencheurs (texte)** : "auto review" / "autoplan" / "run all reviews" / "review this plan automatically" / "make the decisions for me"
- **Auto-invocation** : Oui — Claude propose ce skill quand tu as un plan prêt et veux passer toutes les relectures sans 15-30 questions intermédiaires.

Description détaillée

`/autoplan` lit les fichiers de skill de `/plan-ceo-review`, `/plan-eng-review` et `/plan-devex-review` et les exécute à la suite, dans l'ordre obligatoire CEO → Design → Eng → DX. Chaque phase finit complètement avant que la suivante commence. La qualité de l'analyse reste la même — toutes les sections sont exécutées à fond.

La différence : les questions intermédiaires sont auto-décidées selon six principes. **Choisir la complétude** (livrer la version complète). **Boil the lakes** (corriger tout dans le rayon d'impact si c'est moins d'une journée). **Pragmatisme** (entre deux solutions équivalentes, la plus propre). **DRY** (refuser les doublons). **Explicite plutôt que malin** (10 lignes évidentes plutôt que 200 lignes abstraites). **Biais vers l'action** (avancer plutôt que tergiverser).

Les décisions de goût (« deux approches sont également valables ») sont mises de côté et présentées tout à la fin dans un récap d'approbation. Si Claude et Codex sont tous les deux d'accord pour challenger ta direction (par exemple fusionner deux features que tu voulais séparées), c'est traité spécialement : ta direction reste par défaut, les deux IA doivent défendre le changement.

Une seule commande, plan complètement reviewé en sortie. C'est l'équivalent d'une grosse réunion de plan compressée en une session.

Source

- **Plugin** : `gstack`
- **Nom interne** : `autoplan`
- **Fichier** : `/home/thymon/.claude/skills/gstack/autoplan/SKILL.md`

plan-eng-review

plan-eng-review

> *Catalogue généré le 2026-05-11*

En une phrase

Une relecture de plan en mode « manager d'ingénierie » : Claude verrouille ton architecture, tes flux de données, tes cas limites et ta couverture de tests avant que tu codes.

Quand l'utiliser

- Quand tu as un plan ou un design doc et tu t'apprêtes à commencer à coder
- Quand tu veux t'assurer que les diagrammes, l'architecture et les flux sont solides
- Quand tu veux qu'on traque tous les cas limites avant qu'ils explosent en production
- Après `/plan-ceo-review` (qui s'occupe du « pourquoi »), pour s'occuper du « comment »
- Avant d'attaquer un gros refacto ou un nouveau système

Comment l'invoquer

- **Slash command** : `/plan-eng-review` (à taper dans Claude Code)
- **Voice triggers** : « tech review » · « technical review » · « plan engineering review »
- **Phrases déclencheurs (texte)** : "review the architecture" / "engineering review" / "lock in the plan" / "review architecture"
- **Auto-invocation** : Oui — Claude propose ce skill quand tu as un plan ou design doc et tu t'apprêtes à coder, pour repérer les soucis d'architecture avant l'implémentation.

Description détaillée

`/plan-eng-review` joue le rôle d'un manager d'ingénierie expérimenté. Avant toute relecture, il fait un « scope challenge » : est-ce que du code existant résout déjà ce problème ? Quelle est la version minimale ? Si le plan touche plus de 8 fichiers ou crée plus de 2 nouveaux services, c'est un signal d'alarme qui déclenche une remise en question.

Le skill applique des principes solides : un seul changement à la fois (refacto puis comportement, jamais les deux), DRY, observabilité (logs, métriques, traces), sécurité par défaut, code « bien ingénieré » (ni fragile ni sur-conçu). Il valorise beaucoup les diagrammes ASCII pour les flux de données, machines à états, pipelines, et exige qu'ils restent à jour.

La relecture passe par quatre sections en mode interactif : Architecture, Qualité du code, Tests, Performance. Pour chaque section, jusqu'à huit problèmes principaux sont présentés un par un avec recommandation, pour/contre, et tu décides. Une règle anti-shortcut empêche Claude d'écrire tous les retours dans un fichier de plan sans te les présenter — chaque trouvaille passe par toi.

À la fin tu obtiens un plan blindé prêt à coder, avec les bonnes briques techniques, les bons tests, et les bons garde-fous. Aucun code n'est écrit pendant la session.

Source

- **Plugin** : `gstack`
- **Nom interne** : `plan-eng-review`
- **Fichier** : `/home/thymon/.claude/skills/gstack/plan-eng-review/SKILL.md`

plan-devex-review

plan-devex-review

> *Catalogue généré le 2026-05-11*

En une phrase

Une relecture de plan en mode « expérience développeur » : Claude vérifie que ton API, ta CLI ou ton SDK seront agréables à utiliser pour d'autres devs.

Quand l'utiliser

- Quand tu construis quelque chose que d'autres développeurs vont utiliser (API, CLI, SDK, librairie, framework, plateforme)
- Quand tu veux que ton « time to hello world » (TTHW) soit court : moins de 2 minutes pour un premier essai réussi
- Quand tu veux benchmarker ton approche contre les standards (Stripe, Vercel, Next.js, etc.)
- Quand tu écris la doc d'onboarding et tu veux qu'elle soit irréprochable
- Avant de publier publiquement un outil pour développeurs

Comment l'invoquer

- **Slash command** : `/plan-devex-review` (à taper dans Claude Code)
- **Voice triggers** : « dx review » · « developer experience review » · « devex review » · « devex audit » · « API design review » · « onboarding review »
- **Phrases déclencheurs (texte)** : "DX review" / "developer experience audit" / "devex review" / "API design review"

- **Auto-invocation** : Oui — Claude propose ce skill quand tu as un plan pour un produit destiné aux développeurs (APIs, CLIs, SDKs, librairies, plateformes, docs).

Description détaillée

`/plan-devex-review` adopte le point de vue d'un developer advocate qui a déjà onboardé sur cent outils différents. La DX (Developer Experience), c'est l'UX pour développeurs — mais avec une barre plus haute, parce que tes utilisateurs cuisinent eux-mêmes pour vivre.

Le skill propose trois postures : **DX EXPANSION** (chercher l'avantage compétitif, viser le tier Stripe/Vercel), **DX POLISH** (blinder chaque point de contact), **DX TRIAGE** (corriger seulement les manques critiques). Il évalue ton plan selon huit principes : zéro friction au démarrage, étapes incrémentales, apprendre en faisant, défauts opinionnés avec échappatoires, lutter contre l'incertitude (chaque erreur = problème + cause + correctif), montrer du code en contexte, vitesse comme fonctionnalité, créer des moments magiques.

Sept dimensions sont notées de 0 à 10 : Utilisable, Crédible, Trouvable, Utile, Précieux, Accessible, Désirable. Pour chaque note, le skill explique ce qu'il faudrait pour atteindre 10. Il chronométre aussi le TTHW (Time to Hello World) avec des tiers clairs : champion (<2min), compétitif (2-5min), à améliorer (5-10min), drapeau rouge (>10min).

Le skill explore d'abord les personas de développeurs cibles, écrit un récit d'empathie (« voilà ce que vit Sarah en arrivant »), benchmarke contre les leaders du domaine, puis te présente les améliorations à apporter. La sortie est un plan blindé pour produire une expérience que les développeurs ont envie de partager.

Source

- **Plugin** : `gstack`
- **Nom interne** : `plan-devex-review`
- **Fichier** : `/home/thymon/.claude/skills/gstack/plan-devex-review/SKILL.md`

office-hours

office-hours

> *Catalogue généré le 2026-05-11*

En une phrase

Une séance type « bureau YC » : Claude joue le rôle d'un partenaire qui te pose les bonnes questions avant que tu construises quoi que ce soit.

Quand l'utiliser

- Quand tu as une idée et tu te demandes si ça vaut le coup de la construire
- Quand tu veux brainstormer un projet, un side-project, une feature, un hackathon
- Avant de lancer `/plan-ceo-review` ou `/plan-eng-review`, pour avoir un design doc solide en entrée
- Quand tu n'arrives pas à articuler clairement le problème que tu veux résoudre
- Quand tu te lances dans quelque chose de nouveau et tu veux poser les bases proprement

Comment l'invoquer

- **Slash command** : `/office-hours` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "brainstorm this" / "I have an idea" / "help me think through this" / "office hours" / "is this worth building"
- **Auto-invocation** : Oui — Claude invoque ce skill automatiquement quand tu décris une nouvelle idée de produit, demandes si ça vaut le coup de la construire, ou explores un concept avant d'écrire le moindre code.

Description détaillée

`/office-hours` est une session de brainstorming structurée. Avant tout, le skill te demande ton objectif : startup, intrapreneuriat, hackathon, open source, apprentissage, ou juste pour le fun. Selon ta réponse, il bascule dans l'un des deux modes.

Mode startup applique six questions de diagnostic YC : la demande est-elle réelle (pas juste de l'intérêt poli) ? Quel est le concurrent réel (souvent un tableur + Slack bricolés) ? Quelle spécificité désespérante peux-tu nommer (un vrai client, un vrai cas) ? Quel est le wedge le plus étroit qui peut générer du revenu cette semaine ? As-tu observé directement des utilisateurs ? Le produit tient-il dans 5-10 ans ? Le ton est exigeant : la spécificité est la seule monnaie, l'intérêt n'est pas la demande, les mots de l'utilisateur battent le pitch du fondateur.

Mode builder est plus chaleureux : design thinking, exploration, alternatives, brainstorming créatif pour les projets perso ou d'apprentissage.

Une règle stricte : aucune ligne de code n'est écrite, aucun autre skill d'implémentation n'est invoqué. La sortie est uniquement un design doc structuré (problème, contraintes, approche choisie, alternatives explorées) sauvegardé dans `~/.gstack/projects/<projet>/`. Ce document devient ensuite l'entrée des skills suivants (`/plan-ceo-review`, `/plan-eng-review`).

Source

- **Plugin** : `gstack`
- **Nom interne** : `office-hours`
- **Fichier** : `/home/thymon/.claude/skills/gstack/office-hours/SKILL.md`