Chapitre B — Browser, QA & Dogfooding

9 skills regroupés sous ce thème.

- setup-browser-cookies
- scrape
- pair-agent
- canary
- connect-chrome
- browse
- open-gstack-browser
- qa-only
- qa

setup-browser-cookies

setup-browser-cookies

> *Catalogue généré le 2026-05-11*

En une phrase

Importe les cookies de connexion de ton vrai navigateur Chrome vers le navigateur invisible utilisé par les skills de test, pour qu'il soit déjà connecté à tes comptes.

Quand l'utiliser

- Avant de tester une page qui demande d'être connecté (admin, dashboard, espace membre).
- Quand un skill de QA bute sur une page de login et ne sait pas comment y entrer.
- Pour faire dogfooder ton site comme si tu étais un utilisateur déjà authentifié.
- Quand tu veux éviter de re-taper un mot de passe à chaque session de test.

Comment l'invoquer

- **Slash command** : `/setup-browser-cookies`
- **Phrases déclencheurs (texte)** : "import cookies" / "login to the site" / "authenticate the browser" / "setup authenticated session"
- **Auto-invocation** : Sur demande explicite, ou suggéré par un autre skill quand un test bute sur un mur de connexion.

Description détaillée

Ce skill ouvre une petite interface dans ton navigateur où tu choisis quels domaines (par exemple github.com, ton dashboard, etc.) tu veux importer. Tu cliques sur le "+" à côté du domaine voulu et les cookies de session sont copiés vers le navigateur invisible que gstack utilise pour les tests.

Il détecte tout seul les navigateurs Chromium installés sur ta machine (Chrome, Comet, Brave, etc.). Tu peux aussi lui passer directement un domaine en argument si tu sais ce que tu veux importer, sans passer par l'interface graphique.

Une fois les cookies importés, ils restent disponibles pour toutes les commandes suivantes du navigateur de test. Si tu utilises déjà le mode CDP (le navigateur invisible est branché sur ton vrai navigateur), ce skill détecte le cas et te dit que ce n'est pas nécessaire — tes sessions sont déjà partagées.

Côté sécurité, l'interface n'affiche que les noms de domaines et le nombre de cookies, jamais leur contenu. Sur macOS, la première importation peut déclencher une demande du Trousseau (Keychain) — il faut cliquer sur "Toujours autoriser".

Source

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

scrape

scrape

> *Catalogue généré le 2026-05-11*

En une phrase

Récupère les données affichées sur une page web (texte, prix, liste, etc.) et te les rend en JSON propre, sans avoir à écrire de scraper toi-même.

Quand l'utiliser

- Quand tu veux extraire le contenu d'une page (article, liste de produits, prix, tableau) pour le réutiliser ailleurs.
- Pour récupérer les infos d'une page que tu n'as pas le droit ou le temps de coder une API contre.
- Pour comparer le contenu de deux URLs (par exemple ton site avant/après un déploiement).
- Quand tu poses la question "qu'est-ce qu'il y a sur cette page ?" en cherchant des données précises.

Comment l'invoquer

- **Slash command** : `/scrape`
- **Phrases déclencheurs (texte)** : "scrape" / "get data from" / "pull from" / "extract from" / "what's on" a page
- **Auto-invocation** : Oui — Claude propose ce skill quand tu décris une tâche d'extraction de données depuis le web.

Description détaillée

Ce skill est en lecture seule : il navigue vers une page, regarde le contenu, et retourne un JSON structuré. Il ne clique pas sur des boutons, ne remplit pas de formulaires, ne soumet rien — pour ces flux qui modifient l'état d'un site, tu utiliserais `/automate` à la place.

La première fois que tu lances un scraping avec une nouvelle intention (par exemple "récupère tous les titres d'articles du blog"), le skill prototype le flux en utilisant les primitives du navigateur invisible de gstack. Il analyse la page, repère les éléments à extraire, et retourne le JSON.

Les appels suivants sur la même intention sont automatiquement routés vers une "browser-skill" codifiée (un mini-programme spécialisé qu'il a généré lui-même) qui répond en environ 200 ms. Autrement dit : la première fois c'est lent (le temps de réfléchir à la structure), les fois d'après c'est très rapide.

Ce mécanisme est utile pour des tâches répétitives : tu peux par exemple scraper le statut d'une page de production toutes les heures sans payer le coût d'analyse à chaque fois.

Source

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

pair-agent

pair-agent

> *Catalogue généré le 2026-05-11*

En une phrase

Permet à un autre agent IA (Codex, Cursor, OpenClaw, etc.) de contrôler ton navigateur, dans son propre onglet, avec des permissions limitées et sécurisées.

Quand l'utiliser

- Quand tu veux faire travailler deux agents IA différents sur le même site, sans qu'ils se marchent dessus.
- Pour donner un accès lecture/écriture limité à un agent distant qui doit naviguer (par exemple pour faire du test ou du scraping).
- Quand tu collabores avec un autre agent (OpenClaw, Hermes, etc.) qui a besoin d'un vrai navigateur.
- Pour partager temporairement ta session de navigateur avec quelqu'un d'autre, sans donner tes identifiants.

Comment l'invoquer

- **Slash command** : `/pair-agent`
- **Voice triggers** : « pair agent » · « connect agent » · « share my browser » · « remote browser access »
- **Phrases déclencheurs (texte)** : "pair agent" / "connect agent" / "share browser" / "remote browser" / "let another agent use my browser" / "give browser access"
- **Auto-invocation** : Sur demande explicite uniquement.

Description détaillée

Ce skill génère une clé d'appairage et imprime les instructions exactes que l'autre agent doit suivre pour se connecter à ton navigateur. L'autre agent peut être absolument n'importe quoi qui sait faire un appel HTTP : OpenClaw, Hermes, Codex CLI, Cursor, ou même un script maison.

L'agent distant reçoit son propre onglet dans ton navigateur, avec un accès "scoped" — par défaut, lecture et écriture sur les pages, mais pas les commandes admin du navigateur. Si l'autre agent a besoin de plus, il doit le demander explicitement.

Côté sécurité, c'est costaud : le démon démarre deux écouteurs HTTP. Un local (127.0.0.1) garde toutes les commandes, et un tunnel ngrok ne laisse passer qu'une allowlist très restreinte (`/connect`, `/command` avec un token limité à 26 commandes de pilotage, et `/sidebar-chat`). Les tentatives bloquées sont loggées. Bref : tu peux partager sans flipper.

C'est utile principalement quand tu construis des pipelines multi-agents ou que tu veux qu'un agent externe vienne dépanner sur ton setup sans avoir à dupliquer toute la config de navigateur.

Source

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

canary

canary

> *Catalogue généré le 2026-05-11*

En une phrase

Surveille ton site en production après un déploiement, prend des captures d'écran à intervalles réguliers, et te prévient si quelque chose se casse.

Quand l'utiliser

- Juste après un déploiement, quand tu veux être sûr que rien n'est cassé sans rester collé à ton écran.
- Pour détecter automatiquement les régressions de performance (le site qui devient lent).
- Pour repérer les erreurs JavaScript dans la console qui sont apparues après ta mise en ligne.
- Quand tu veux surveiller une page sensible pendant quelques heures sans intervention humaine.

Comment l'invoquer

- **Slash command** : `/canary`
- **Phrases déclencheurs (texte)** : "monitor deploy" / "canary" / "post-deploy check" / "watch production" / "verify deploy" / "monitor after deploy" / "canary check"
- **Auto-invocation** : Sur demande explicite, ou suggéré juste après un déploiement effectué via `/ship` ou `/land-and-deploy`.

Description détaillée

Le terme "canary" vient des canaris qu'on emmenait dans les mines de charbon : si l'oiseau tombait, on savait qu'il y avait un problème dans l'air. Ici, le canary, c'est ce skill qui veille sur ton site en production.

Concrètement, il ouvre le démon du navigateur invisible de gstack et lance des sondages réguliers sur ton site live. À chaque passage, il prend une capture d'écran, regarde la console pour repérer les erreurs JS, mesure le temps de chargement, et compare tout ça à des références prises avant le déploiement (les "baselines").

Si quelque chose s'écarte trop de la baseline — une erreur en console qui n'était pas là, une page qui met soudainement 3 secondes au lieu de 500 ms, un élément clé qui a disparu — il t'alerte.

Tu peux le lancer juste après `/ship` ou `/land-and-deploy` pour avoir l'esprit tranquille pendant que tu fais autre chose. C'est une sorte de "monitoring léger maison" qui ne remplace pas un vrai outil de monitoring pro, mais qui rattrape déjà 80% des régressions visibles en post-deploy.

Source

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

connect-chrome

connect-chrome

> *Catalogue généré le 2026-05-11*

En une phrase

Lance une fenêtre de Chromium pilotée par l'IA, visible à l'écran, avec une extension de sidebar qui affiche en temps réel tout ce que Claude fait dans le navigateur.

Quand l'utiliser

- Quand tu veux VOIR Claude utiliser un navigateur (au lieu du mode invisible classique).
- Pour tester un site en regardant chaque clic, chaque saisie, chaque navigation.
- Quand tu veux discuter avec Claude pendant qu'il navigue, via la chat intégrée dans la sidebar.
- Pour contourner les protections anti-bot de certains sites (le mode stealth est intégré).
- Voice : quand tu dis "montre-moi le navigateur" pendant que tu lui parles.

Comment l'invoquer

- **Slash command** : `/connect-chrome`
- **Voice triggers** : « show me the browser »
- **Phrases déclencheurs (texte)** : "open gstack browser" / "launch browser" / "connect chrome" / "open chrome" / "real browser" / "launch chrome" / "side panel" / "control my browser"
- **Auto-invocation** : Sur demande explicite uniquement.

Description détaillée

Ce skill est un alias de `/open-gstack-browser` (le nom historique). Il ouvre une fenêtre Chromium classique, à l'écran, que Claude peut piloter exactement comme s'il s'agissait du navigateur invisible — sauf que là, tu vois tout en direct.

L'extension de sidebar est pré-installée à l'intérieur. Elle affiche un fil d'activité (chaque action de Claude apparaît en temps réel), un terminal PTY connecté à ta session Claude pour échanger avec lui pendant qu'il navigue, et un inspecteur CSS pour examiner les éléments visuellement.

Sous le capot, c'est le même navigateur que `/browse` utilise quand il est en mode invisible — mêmes commandes, même comportement, mais visible. Le mode "stealth" anti-bot est intégré : les sites qui détectent les navigateurs automatisés (Cloudflare, certains shops) ont beaucoup plus de mal à te repérer.

C'est l'outil idéal quand tu fais du debug visuel ou que tu veux apprendre comment Claude raisonne en regardant ses actions se dérouler à l'écran.

Source

- **Plugin** : `gstack`
- **Nom interne** : `open-gstack-browser` (le dossier `connect-chrome` est un alias)
- **Fichier** : `/home/thymon/.claude/skills/gstack/connect-chrome/SKILL.md`

browse

browse

> *Catalogue généré le 2026-05-11*

En une phrase

Donne à Claude un navigateur invisible et rapide pour ouvrir une page, cliquer, remplir des champs, prendre des captures, et tester ton site comme un utilisateur.

Quand l'utiliser

- Quand tu veux que Claude teste lui-même une fonctionnalité après l'avoir codée.
- Pour vérifier qu'un déploiement n'a rien cassé (smoke test rapide).
- Pour prendre une capture d'écran d'un bug et la joindre à un rapport.
- Pour tester comment ton site rend sur différentes tailles d'écran (responsive).
- Pour faire dogfooder un parcours utilisateur de bout en bout (login, formulaire, paiement…).

Comment l'invoquer

- **Slash command** : `/browse`
- **Phrases déclencheurs (texte)** : "open in browser" / "test the site" / "take a screenshot" / "dogfood this" / "browse a page" / "headless browser"
- **Auto-invocation** : Sur demande explicite, et utilisé en interne par presque tous les autres skills QA et design.

Description détaillée

C'est la brique de base sur laquelle s'appuient tous les autres skills de QA et de design de gstack. C'est un navigateur Chromium "headless" (sans interface graphique visible) basé sur Playwright, qui répond en environ 100 ms par commande.

Il sait tout faire : naviguer vers une URL, cliquer sur un élément, remplir un formulaire, gérer les dialogues (alert/confirm), faire des assertions ("ce bouton existe", "ce texte est présent"), comparer l'état avant/après une action, prendre des captures d'écran annotées, tester en mode mobile ou desktop, et gérer les uploads de fichiers.

Pour Thymon, l'intérêt n'est pas tant de le piloter à la main que de comprendre qu'il existe : quand un skill comme `/qa`, `/design-review` ou `/canary` parle de "tester ton site", c'est `browse` qui fait le boulot en coulisse.

Tu peux aussi le lancer directement quand tu as besoin d'une action ponctuelle — par exemple "prends une capture de ma page d'accueil sur iPhone 12". Pour la version VISIBLE (où tu vois la fenêtre), utilise `/open-gstack-browser` ou `/connect-chrome`.

Source

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

open-gstack-browser

open-gstack-browser

> *Catalogue généré le 2026-05-11*

En une phrase

Lance la fenêtre visible du navigateur Chromium piloté par Claude, avec l'extension sidebar baked-in pour voir tout ce qu'il fait en direct.

Quand l'utiliser

- Quand tu veux voir Claude utiliser un navigateur à l'écran (au lieu du mode invisible).
- Pour suivre en direct un parcours de test, clic par clic.
- Quand tu veux discuter avec Claude pendant qu'il navigue, via la chat de la sidebar.
- Pour contourner des protections anti-bot — le mode stealth est intégré dès l'ouverture.
- Voice : quand tu dis "montre-moi le navigateur" pendant une session vocale.

Comment l'invoquer

- **Slash command** : `/open-gstack-browser`
- **Voice triggers** : « show me the browser »
- **Phrases déclencheurs (texte)** : "open gstack browser" / "launch chromium" / "open chrome" / "real browser" / "launch browser" / "side panel" / "control my browser"
- **Auto-invocation** : Sur demande explicite uniquement.

Description détaillée

C'est le skill canonique pour ouvrir le navigateur de gstack en mode visible. `/connect-chrome` est juste un alias plus court qui pointe vers ce même skill — le nom officiel et le dossier de code, c'est `open-gstack-browser`.

Quand tu le lances, une vraie fenêtre Chromium s'ouvre. Claude la pilote exactement comme il pilote le mode invisible (`/browse`), mais cette fois tu vois tout. À côté de la zone web, une sidebar te montre : un fil d'activité avec chaque action exécutée, un terminal PTY interactif qui te connecte à ta session Claude pour lui parler sans quitter le navigateur, et un inspecteur CSS pour examiner les éléments en cours.

Le mode "anti-bot stealth" est activé par défaut. Concrètement, ça veut dire que les sites qui détectent les navigateurs automatisés (Cloudflare, certains marketplaces, certains paywalls) ont beaucoup plus de mal à te bloquer.

C'est l'outil parfait pour le debug visuel, l'apprentissage (regarder comment Claude raisonne), et les sites qui résistent au mode headless classique.

Source

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

qa-only

qa-only

> *Catalogue généré le 2026-05-11*

En une phrase

Teste ton site et te rend un rapport de bugs détaillé avec captures et étapes pour reproduire — mais ne touche jamais au code, jamais.

Quand l'utiliser

- Quand tu veux UN AVIS sur ce qui ne va pas, sans que Claude commence à "réparer" des trucs tout seul.
- Pour générer un rapport de bugs propre à transmettre à un collègue ou à reprendre toi-même plus tard.
- Avant un point de revue où tu veux savoir l'état réel du site, sans pollution par des modifs.
- Quand tu veux une note de "santé" du site à un moment T pour comparer plus tard.

Comment l'invoquer

- **Slash command** : `/qa-only`
- **Voice triggers** : « bug report » · « just check for bugs »
- **Phrases déclencheurs (texte)** : "just report bugs" / "qa report only" / "test but don't fix"
- **Auto-invocation** : Oui — Claude suggère ce skill quand tu demandes un rapport de bugs sans toucher au code.

Description détaillée

C'est la version "audit pur" de `/qa`. Elle exécute exactement la même batterie de tests systématiques sur ton application web — navigation, formulaires, états vides, responsive, console errors, performance — mais avec une règle absolue : interdiction de modifier le code.

Le résultat, c'est un rapport structuré qui contient un score de santé global, la liste des problèmes classés par criticité (critical / high / medium / cosmetic), des captures d'écran pour chaque bug, et les étapes exactes pour le reproduire. Tu peux ensuite décider toi-même ce que tu veux corriger, ou refiler le rapport à quelqu'un d'autre.

Le contraste avec `/qa` est important : `/qa` lance la boucle complète "tester → corriger → vérifier" et ferme les bugs au fur et à mesure. `qa-only` s'arrête au "tester" et ne fait rien d'autre. Tu choisis selon que tu veux l'autonomie complète ou la simple constatation.

C'est aussi un bon outil pour figer l'état d'un site dans le temps : tu lances `qa-only` aujourd'hui, tu fais des modifs pendant deux semaines, tu le relances, tu compares les deux rapports.

Source

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

qa

qa

> *Catalogue généré le 2026-05-11*

En une phrase

Teste systématiquement ton site, trouve les bugs, et les corrige un par un dans le code en commit propre, jusqu'à ce que tout fonctionne.

Quand l'utiliser

- Quand tu viens de finir une feature et tu veux savoir si "ça marche vraiment".
- Pour un check complet avant de mettre en ligne — formulaires, navigation, responsive, edge cases.
- Quand tu poses la question "est-ce que ça marche ?" et tu veux une vraie réponse, pas une intuition.
- Quand tu acceptes que Claude corrige les bugs trouvés lui-même (sinon préfère `/qa-only`).

Comment l'invoquer

- **Slash command** : `/qa`
- **Voice triggers** : « quality check » · « test the app » · « run QA »
- **Phrases déclencheurs (texte)** : "qa" / "QA" / "test this site" / "find bugs" / "test and fix" / "fix what's broken"
- **Auto-invocation** : Oui — Claude suggère ce skill quand tu dis qu'une feature est prête à tester ou quand tu poses la question "est-ce que ça marche ?".

Description détaillée

C'est le couteau suisse du test : il fait à la fois l'audit ET la réparation. La boucle complète : tester systématiquement, trouver les bugs, corriger dans le code source, commiter atomiquement, et re-tester pour confirmer que c'est bien réglé. Tout ça en autonomie.

Tu choisis le niveau d'exhaustivité :

- **Quick** : seulement les bugs critiques et high (ce qui empêche le site de marcher).
- **Standard** : critical + high + medium (la couverture habituelle).
- **Exhaustive** : tout, jusqu'aux soucis cosmétiques (alignement, micro-typographie).

À la fin, tu reçois un résumé avec deux scores de santé — celui d'avant la passe et celui d'après — la preuve des corrections (captures avant/après pour chaque fix), et un verdict "prêt à shipper" ou "il reste des soucis bloquants".

À ne pas confondre avec `/qa-only`, qui fait le même audit mais s'arrête à la simple constatation sans toucher au code. Utilise `/qa` quand tu fais confiance à Claude pour corriger, et `/qa-only` quand tu veux garder le contrôle sur les modifs.

Source

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