

DICKO SOKA · LE LIVRE

De l'idée au produit

Le parcours complet, expliqué à qui ne programme pas

Mohamed Nassirou SOUMANOU

Deuxième édition

Dicko Soka

De l'idée au produit

Le parcours complet, expliqué à qui ne programme pas

Ils vous vendent le cheval.
Nous vous apprenons à tenir la bride.

Mohamed Nassirou SOUMANOU
Fondateur de Dicko Industries

Deuxième édition
Décrit le harnais Dicko Soka 3.169.64, fabriquée le 24 septembre 2026
Édité par Dicko Industries

Je ne sais pas écrire une ligne de code

Il faut le dire avant tout le reste, parce que c'est la seule chose qui donne du poids à ce livre : je ne suis pas développeur. Je n'ai ni diplôme ni carrière dans l'informatique. Je ne sais pas écrire une ligne de code.

Je suis diplômé en Management de Terminaux Portuaire auprès de la Lloyds Maritime Academy de Londres en Angleterre et douze ans de métier dans les opérations, la logistique et les terminaux portuaires en Afrique de l'Ouest. Ce que je sais faire, c'est monter une opération qui n'existe pas encore, avec une équipe qui ne la connaît pas encore, dans un délai qui ne laisse pas de marge.

Et depuis l'université, j'ai développé une passion pour l'ordinateur, grâce notamment à AIESEC et au Campus Numérique Francophone de l'UL (là où j'ai découvert les logiciels libres et le système d'exploitation Ubuntu/Linux).
Pas une compétence : une **passion**. La distinction compte, parce qu'elle décrit exactement la position de la plupart des gens qui liront ces pages : curieux de la machine, sans jamais avoir eu le temps ni la raison d'en apprendre le langage.

Ce livre est donc écrit depuis l'autre côté du mur. Pas par quelqu'un qui vulgarise ce qu'il maîtrise depuis toujours, mais par quelqu'un qui a appris récemment, en pratiquant, et qui se souvient encore précisément de ce qui était incompréhensible la veille.

> **Je n'ai pas appris à coder. J'ai appris à tenir la bride.**

Ce que douze ans d'opérations m'ont appris

On m'a confié plusieurs fois une chose qui n'existait pas.

En janvier 2026, un port sec à ouvrir près de Freetown en vingt-cinq jours : à mon arrivée, notre partenaire dégageait encore le terrain. En 2022, une opération d'import, stockage et exportation du coton à Adétikopé : la plateforme n'en avait jamais manutentionné, moi non plus, et le seul document qu'on m'a remis était la spécification d'empilage de l'assureur. Avant cela, j'ai eu à monter les opérations du site de dépotage des conteneurs ainsi que les opérations des entrepôts, après avoir construit la fonction Process Excellence sur un terminal à conteneurs de grande envergure à Lomé, au Togo.

À chaque fois, la même méthode : écouter avant de décider, définir le périmètre avant d'agir, verrouiller les règles non négociables *avant* la première opération, puis corriger par cycles.

En découvrant le harnais de pilotage de l'IA, je n'ai donc pas découvert une idée nouvelle : **j'ai reconnu la mienne, appliquée à un autre métier.** Un harnais logiciel est une procédure d'exploitation, avec son ordre des étapes, ses règles verrouillées d'avance et ses points de contrôle. C'est pour cela que j'ai pu l'étendre.

Novembre 2022

En novembre 2022, l'intelligence artificielle conversationnelle est mise à la disposition du grand public. J'ai été captivé immédiatement, et je ne me suis plus arrêté depuis : en autodidacte, et surtout en pratiquant, la seule manière d'apprendre un outil dont la documentation vieillit plus vite qu'on ne la lit.

La première fois que j'ai utilisé Claude, j'ai trouvé autre chose. Étant Manager, ce qui m'intéressait n'était pas la performance technique : c'était la capacité à raisonner sur un problème mal posé, à structurer ce qui arrive en désordre, à me contredire quand j'avais tort. Un Manager passe ses journées là-dedans. J'ai donc approfondi cet outil-là plutôt qu'un autre, et je n'ai pas cessé depuis.

Claude m'offrait la possibilité d'approfondir certains sujets, de faire du brainstorming et surtout de produire des livrables en word ou Excel de qualité professionnel. Mais quelque chose manquait, et je ne savais pas le nommer. Pour aller plus loin, surtout avec le code, il devenait évident que je n'avais pas les compétences qu'il fallait. J'ai essayé de produire une application métier (dans le domaine de la logistique) avec Claude Code. Cela m'a consumé beaucoup de temps et d'énergie sans que l'application ne prenne réellement forme. Mes abonnements à Claude Pro puis Max, n'étaient donc pas rentabilisés comme je le souhaitais.

Ce n'était pas un problème d'intelligence de la machine. C'était un problème d'**ordre et de méthode**, et c'est exactement mon métier, sauf que je ne l'avais pas encore vu.

> **Je savais ce que je voulais construire. Mais je n'avais pas l'outil qui me permettait de faire travailler la machine dans l'ordre qu'il faut.**

Juillet 2026 : le mot « harnais »

Début juillet 2026, je tombe sur une vidéo YouTube. Un jeune développeur français, **Flavien Chervet**, y est interviewé, et il emploie un mot que je n'avais pas entendu dans ce sens : le ***harnais***.

Son constat était celui que je n'arrivais pas à formuler : le harnais est ce qui manquait aux non-techniciens pour obtenir de véritables résultats de l'IA. Pas un modèle plus puissant, pas une meilleure formulation des demandes, mais une couche au-dessus, qui sait quoi faire et dans quel ordre. Il expliquait en avoir mis un en place, et que ses étudiants non-techniciens menaient avec lui des projets de bout en bout.

Il annonçait qu'il l'ouvrirait plus tard en **open source** (logiciel dont le code est mis à disposition du public), et j'ai attendu cette ouverture.

Quand il l'a ouvert, je l'ai pris, je l'ai étudié, et j'ai commencé à l'adapter aux réalités de l'Afrique de l'Ouest en général et de la zone UEMOA en particulier.

ATTRIBUTION

Dicko Soka descend d'un harnais ouvert en open source par Flavien Chervet. L'idée du harnais, et la première mise en œuvre dont nous sommes partis, sont les siennes. Ce livre décrit ce que nous en avons fait ; il ne prétend pas à l'invention de ce dont il hérite.

Ce que nous y avons ajouté

Un harnais conçu ailleurs suppose un marché qui n'est pas le nôtre, et la première divergence est le paiement.

Là où le harnais d'origine parlait d'intégrer **Stripe**, j'ai ajouté **FedaPay**, qui encaisse par mobile money. Ce n'est pas une préférence technique : c'est ainsi que nos clients paient. Un produit ouest-africain qui n'accepte que la carte refuse la majorité de son marché.

Puis d'autres commandes, chacune née d'un manque rencontré (la liste ci-dessous n'est pas exhaustive de nos commandes propres mais elle est à titre indicatif) :

`/pain-point-finder`	Chercher les points de douleur documentés d'un marché, au lieu de partir d'une intuition.
`/bmc`	Schématiser le modèle économique, parce qu'une idée sans modèle est un passe-temps.
`/generate` · `/montage`	Produire et assembler les visuels, parce qu'un bon produit invisible ne se vend pas.

Et d'autres encore, plus une réécriture continue de ce qui existait. Aujourd'hui, **Dicko Soka est à environ 60 % notre travail propre.**

LA PREUVE

Dicko Soka m'a permis de concevoir **TransiFa Docs** sans aucun développeur. Le service prépare les déclarations en douane au Togo, avec des professionnels qui le paient au dossier.

Pourquoi je m'y investis

J'ai perçu la portée stratégique de l'intelligence artificielle, et c'est la raison de cet investissement, et surtout pour partager mes expériences et augmenter le nombre de jeunes Africains qui se forment dans ce domaine d'avenir.

Nous pouvons réaliser énormément avec cette technologie. Elle peut nous permettre de jeter des ponts dans certains secteurs, et d'emprunter des raccourcis pour réduire notre écart technologique avec les autres nations, en tirant profit notamment des modèles ouverts, DeepSeek, Qwen, GLM et les suivants.

Mais je reste lucide. Personne au monde ne mettra à notre disposition sa technologie la plus avancée. Il nous revient, en tant qu'Africains, de faire le travail nécessaire pour développer nos propres technologies, en puisant dans notre riche tradition et notre culture, dont le paradigme demeure assez large pour concevoir des technologies que personne d'autre au monde ne peut développer.

C'est une conviction personnelle. Ce livre en est la première pierre : ce que j'ai appris une fois ne doit plus être réappris par le suivant.

Mohamed Nassirou SOUMANOU
Lomé, août 2026

> **Soka, c'est la bride. En éwé, sɔ́ veut dire cheval et ka veut dire corde.**

LA MAISON

Dicko Industries

Trois produits, une même conviction : l'outil doit porter le savoir-faire, pas seulement l'exécution. Ce qu'un professionnel apprend à ses dépens doit devenir une règle que le produit applique de lui-même, qu'il s'agisse d'un harnais logiciel, d'une déclaration en douane ou d'une clé à choc.

Dicko Soka

CE LIVRE

Le harnais, et la méthode qu'il porte : de l'idée au premier client qui paie, en seize étapes ordonnées, sans écrire une ligne de code. Transmis aussi en séances de formation à Lomé et en ligne, autour d'une communauté de bâtisseurs.

dickoindustries.com/soka

TransiFa Docs

EN SERVICE

La déclaration en douane juste du premier coup, au Togo : classement SH assisté sur le tarif togolais, le droit de douane, la valeur CAF prête, la minute et le fichier XML à télécharger et importer dans SYDONIA World. Mille francs CFA le dossier, par mobile money. **Conçu avec Dicko Soka, sans aucun développeur.**

test.transifa.com

Dicko Tools

BOUTIQUE EN LIGNE EN CONSTRUCTION

L'outillage automobile professionnel à Lomé et dans la CEDEAO : crics, extracteurs, outils moteur, diagnostic multimarque. Livraison Lomé en vingt-quatre heures, paiement mobile money, conseil par WhatsApp.

dickotools.com

Ce livre ne traite que du premier. Le deuxième en est né, et c'est la meilleure preuve que la méthode tient : un produit vendu, conçu sans développeur, par l'auteur de ces pages.

Sommaire

	Préface de l'auteur	3
	La maison	7
	Ce livre, et à qui il parle	9
	Ce qui a changé depuis la première édition	11
01	Le principe	15
02	Le parcours en entier	17
03	Si votre projet n'est pas numérique	20
04	Avant de construire	23
05	Ce qui tourne sous le capot	25
06	Le démarrage du projet	28
07	Les outils du quotidien	31
08	Ajouter une capacité	34
09	Mettre en ligne, en trois temps	37
10	Trouver des clients	40
11	Bien décrire ce qu'on veut	42
12	Travailler avec la machine	44
	Annexe · Le glossaire du projet	47
	Pour finir	49
	Par où commencer	50
	Colophon	51

OUVERTURE

Ce livre, et à qui il parle

Vous avez une idée. Peut-être une gêne quotidienne dans votre métier, peut-être un service que personne n'offre autour de vous. Ce qui vous manque n'est pas l'idée : c'est le chemin entre elle et un produit que des gens utilisent et paient.

Ce livre décrit ce chemin. Il ne vous apprend pas à programmer, et c'est délibéré - le harnais Dicko Soka existe précisément pour que vous n'ayez pas à le faire. Il vous apprend **à quoi sert chaque étape, dans quel ordre, et pourquoi cet ordre-là**.

Et si votre projet n'est pas numérique, ne refermez pas ce livre. Une bonne part du parcours ne produit pas une ligne de code. Un atelier, un commerce, une auto-école se conduisent par les mêmes étapes de réflexion qu'une application, parce que ce que le harnais structure n'est pas la construction : c'est la réflexion. Le chapitre 3 est écrit pour vous.

Ce que ce livre n'est pas

Ce n'est pas un manuel qui vous dit où cliquer. Les écrans changent, les manuels avec captures vieillissent en quelques semaines, et un lecteur qui suit une capture périmée conclut que l'outil est cassé alors que c'est le livre qui l'est.

Ce livre est **conceptuel**. Il explique le raisonnement derrière chaque étape, celui qui reste vrai quand un bouton se déplace. Quand vous saurez POURQUOI on isole une base de données avant de tester, la commande exacte sera la partie facile - et l'outil vous la donnera lui-même, puisqu'il est fait pour ça.

Le chiffre qui explique la méthode de ce livre

Ce livre décrit la version **3.169.64** du harnais, qui comprend **69 commandes que vous tapez** et **43 rouages internes** travaillant sans que vous les appeliez, soit **112 en tout**. Le parcours qu'il vous fait suivre compte **16 étapes**.

Ces nombres ne sont pas écrits à la main dans ce livre. Ils sont **lus dans le harnais au moment où le livre est fabriqué** - et c'est une leçon payée. La page de vente du produit a un jour annoncé un total de commandes ; il était faux **vingt heures plus tard**, une

livraison ayant ajouté une commande entre-temps. Un livre imprimé avec un chiffre recopié serait faux avant même d'être téléchargé, et cette fois chez ses lecteurs, sans rattrapage possible.

Alors chaque fait de ce livre - la liste des étapes, leur ordre, les noms des commandes - est relu dans le produit à chaque fabrication. Si vous lisez un chiffre ici, il était vrai pour la version nommée sur la couverture. **Vérifiez toujours cette version**, en première page et dans le colophon en fin d'ouvrage : c'est la date de péremption de ce livre.

Cette édition

C'est la deuxième édition de ce livre. La première décrivait la version 3.86.0 du harnais. Si vous l'avez lue, ou si vous utilisez déjà le harnais, le chapitre qui suit celui-ci dit ce qui a changé depuis, et où chaque nouveauté est reprise dans le livre.

Comment le lire

Les chapitres numérotés suivent le parcours dans son ordre réel, de l'idée non validée jusqu'aux premiers clients. Vous pouvez les lire d'affilée, ou ouvrir directement celui de l'étape où vous êtes.

Une annexe, en fin d'ouvrage, décrit le glossaire du projet : le fichier où s'écrivent les mots de VOTRE métier, et que le cahier des charges et les explications relisent pour parler avec vos mots.

Une seule chose à retenir si vous ne retenez rien d'autre : quand vous ne savez pas quoi faire, le harnais répond à cette question lui-même. La commande `/next` regarde où vous en êtes et nomme l'étape suivante avec sa raison. C'est la seule commande qu'il faut mémoriser.

DEUXIÈME ÉDITION

Ce qui a changé depuis la première édition

La première édition de ce livre, parue en août 2026, décrivait la version 3.86.0 du harnais. Celle-ci décrit la version **3.169.64**. Entre les deux, le harnais a été livré presque chaque jour, et une grande part de ce travail ne se voit pas : il renforce l'intérieur - ce qui vérifie, ce qui refuse, ce qui protège.

Ce chapitre ne retient que ce que vous **remarquerez en l'utilisant**. La plupart de ces nouveautés sont développées plus loin, à leur place dans le parcours : le chapitre qui les reprend est nommé à la fin de chaque section. Si vous découvrez le harnais avec ce livre, vous pouvez passer directement au chapitre 1.

Au moment où ce livre est fabriqué, le parcours est celui de la première édition : mêmes étapes, même ordre, mêmes raisons. Le tableau du chapitre 2, relu dans le harnais à chaque fabrication, fait foi. Ce qui a changé, c'est ce que chaque étape fait pour vous.

De nouvelles commandes

`/diagnose`, quand quelque chose ne marche pas : vous donnez trois phrases, il vérifie le reste lui-même et dit franchement ce qui reste inconnu.

`/explain`, quand vous n'avez pas compris : un mot, un message, une erreur, redits simplement, avec les mots de votre projet.

`/optimize`, quand un palier gratuit approche de sa limite : il cherche dans votre code ce qui le consomme, et ne touche à rien sans votre accord.

`/add-ai`, pour mettre de l'intelligence artificielle dans votre produit, avec un devis avant la première ligne de code et un plafond de dépense.

`/add-api-guard`, pour que les accès techniques de votre application soient fermés par défaut, y compris entre deux clients connectés.

Voir « Les outils du quotidien », « Ajouter une capacité » et « Travailler avec la machine ».

Choisir en voyant, pas sur des mots

`/design` ne vous demande plus de choisir une identité sur des adjectifs ou des codes couleur. Il construit trois versions de votre page d'accueil, chacune dans une direction visuelle différente, dans votre projet et sur votre ordinateur - personne d'autre ne les voit - et vous choisissez en les regardant. Avant même la construction, `/spec` peut vous proposer une maquette jetable dans le même esprit.

Voir « Avant de construire » et « Les outils du quotidien ».

Un cahier des charges qui se défend

Le cahier des charges sépare ce qui reste **à préciser** de ce que vous avez mis **hors périmètre**, avec la raison. Le harnais ne construit jamais ce que vous avez écarté, et il vous repose les points ouverts avant de les construire.

Voir « Avant de construire » et « Bien décrire ce qu'on veut ».

Un projet qui naît vérifié

Un projet neuf naît avec ses tests automatiques, et un projet plus ancien les reçoit lors de l'audit de sécurité. Avant la mise en ligne, un test qui échoue bloque - et un test qui n'a pas pu tourner bloque aussi, parce que rien n'a alors vérifié le code.

Voir « Le démarrage du projet » et « Mettre en ligne, en trois temps ».

L'argent, plus sûr

Avec FedaPay, dans un projet qui a sa base de données, **chaque paiement est enregistré**, et la page de retour affiche le statut confirmé par FedaPay, jamais celui qu'annonce le lien par lequel le client revient. Le passage en paiements réels exige alors la preuve qu'un paiement d'essai a réellement abouti.

Avec Stripe, les clés réelles ne vont plus qu'en production : votre site de test ne peut plus encaisser de vraies cartes. Et revenir du mode réel au mode test est devenu une étape guidée, qui remet réellement votre site public en mode test.

Pour les deux, une clé - le mot de passe d'accès que vous remet le service de paiement - collée avec un espace, un caractère invisible ou du mauvais mode est arrêtée avant de partir, avec un message qui dit quoi corriger. Et relancer la commande sur un site qui encaisse déjà en réel ne vous propose plus de refaire le passage : le harnais lit le mode de votre site en ligne, pas celui de votre ordinateur, et vous pose la question s'il ne peut pas le lire.

Voir « Ajouter une capacité ».

Le mobile, plus loin

Sur un projet d'application mobile, la base de données et son isolement fonctionnent désormais comme pour un site. FedaPay encaisse dans l'application, et ce chemin a été éprouvé par un paiement d'essai depuis une application Android installée. Stripe sur mobile est écrit, mais n'a encore jamais été éprouvé : la commande vous le dit avant de commencer.

Une précision, aussi. La première édition présentait l'encaissement sur le web comme la réponse toute faite pour le contenu numérique vendu dans une application - un abonnement, des crédits - sans rien à arbitrer. C'est trop simple : les boutiques d'applications encadrent de près cette voie, et `/add-stripe` refuse désormais de câbler un paiement dans l'application pour ce contenu. Si votre modèle repose là-dessus, lisez « Le démarrage du projet » avant de construire.

L'intelligence de votre produit, à prix maîtrisé

Ce que **votre produit** consomme chaque mois en intelligence artificielle part par défaut sur des modèles ouverts, bien moins chers ; Claude reste proposé comme option, avec son prix du jour. Un agent créé par le harnais naît fermé : il n'envoie d'email qu'aux adresses que vous avez déclarées, et n'envoie de données vers aucun site que vous n'avez pas déclaré.

Voir « Ce qui tourne sous le capot » et « Ajouter une capacité ».

La production reste à vous

Publier votre travail en production demande votre accord, **à tous les niveaux d'autonomie**, y compris le plus permissif, où seuls quelques redéploiements techniques peuvent partir seuls, et sous conditions. La liste écrite des gestes sensibles, que le harnais consulte avant d'agir au lieu de se fier à sa mémoire, existait déjà ; elle est désormais consultée par bien plus de commandes.

Et pour deux gestes sans retour - écrire dans la base de production, changer le marché d'un projet (pays, monnaie, fournisseur de paiement) - la confirmation se donne dans votre propre terminal, hors de la conversation : l'assistant ne peut pas la donner par mégarde à votre place.

Sous Antigravity, les règles du harnais, ses lignes rouges comprises, s'appliquent désormais dans tous vos projets, dès que vous avez relancé `/start` depuis Antigravity.

Voir « Le principe », « Mettre en ligne, en trois temps » et « Travailler avec la machine ».

Ce qui ne casse plus

Quelques gestes du quotidien qui échouaient, parfois en silence, et qui tiennent maintenant :

- changer de nom de domaine met à jour les paiements et les services de CE projet, jamais ceux de vos autres projets ;
- envoyer un réglage vers l'hébergeur n'efface plus celui des autres environnements : la production ne peut plus perdre sa base de cette façon ;
- une connexion à l'hébergeur qui expire se renouvelle seule, au lieu de tout bloquer ;
- remplacer une clé compromise atteint enfin les serveurs de vos agents, pour les clés que la commande sait y chercher ;
- créer une adresse email vérifie d'abord la messagerie déjà en place sur le domaine, pour ne pas couper votre courrier ;
- une nouvelle image d'aperçu peut enfin remplacer l'ancienne quand on partage vos liens sur WhatsApp ou Facebook ;
- supprimer un projet respecte ce que vous avez demandé de garder.

Ce que vous verrez aussi

Avant de vous remettre une affiche ou un message qui annonce une date, un prix, un lieu ou un contact, le harnais relit ce fait sur votre page en ligne ; s'il n'y est plus, le visuel attend votre décision. Voir « Trouver des clients ».

Sur vos bases de travail et de test aussi, une instruction qui efface une table, ou toutes ses lignes, est d'abord arrêtée et attend votre accord. Voir « Ajouter une capacité ».

Quand une étape recommande un modèle plus capable, le harnais s'arrête, au niveau d'autonomie par défaut, et attend votre réponse. La première édition disait qu'il poursuivait après vous l'avoir signalé : ce n'était pas le cas. Voir « Travailler avec la machine ».

Sous Claude Code, une séance ouverte avant une mise à jour vous prévient quand elle vous sert encore l'ancienne version d'une commande. Même chapitre.

Un projet qui se souvient

Votre projet tient un **registre de ses décisions** - celles qu'on ne veut pas rejouer - que plusieurs commandes relisent avant de vous proposer le contraire. Il garde aussi ses **mots**, dans un glossaire décrit en annexe. Et `/prof` retient ce que vous avez compris d'une séance à l'autre, mais seulement sur preuve.

Voir « Bien décrire ce qu'on veut », « Travailler avec la machine » et l'annexe.

CHAPITRE 01

Le principe

Ce que vous décrivez, la machine le construit

Vous écrivez ce que vous voulez, en français, dans une conversation. La machine écrit le code, crée la base de données, met le site en ligne. Vous ne relisez pas le code : vous regardez le résultat et vous dites ce qui ne va pas.

C'est un renversement, et il mérite d'être nommé : **la compétence rare n'est plus d'écrire le code, c'est de savoir ce qu'on veut**. Décrire précisément un besoin, reconnaître un résultat correct, arbitrer entre deux options - c'est ce que vous apportez, et aucun outil ne le fait à votre place.

Pourquoi un harnais, et pas seulement un assistant

Un assistant seul répond à ce qu'on lui demande. Il le fait bien, et c'est exactement le problème : si vous ne savez pas qu'il faut isoler une base de données avant de tester, il ne le fera pas, et rien ne vous dira que vous venez de tester sur les données réelles de vos clients.

Le harnais est la couche qui **sait ce qu'il faut faire même quand vous ne le demandez pas**. Il connaît l'ordre des étapes, les pièges du marché où vous vendez, les vérifications obligatoires avant une mise en ligne. Ses 112 commandes ne sont pas 112 fonctionnalités : ce sont 112 décisions déjà prises, testées, et qui n'ont plus à être reprises à chaque projet.

Trois règles qu'il ne négocie pas

Ces règles ne sont pas des préférences, ce sont des lignes rouges. Aucun réglage d'autonomie ne les lève.

Aucun mouvement d'argent réel sans votre accord explicite. Pas de paiement déclenché, pas de bascule d'un système de paiement en mode réel, sans que vous ayez dit oui pour cette opération précise.

Aucune suppression de ressource sans votre accord explicite. Supprimer une base, un projet, un dépôt : jamais en votre absence, jamais par déduction.

Aucune écriture dans la base de production sans votre accord explicite. Et ce n'est pas une politesse : par défaut, les instructions envoyées à votre base partent en lecture seule, et c'est la base elle-même qui refuse d'écrire. Une garde qu'on peut contourner par une

formulation habile n'est pas une garde.

Et pour ce geste-là, l'accord ne se donne pas dans la conversation. L'outil vous donne une ligne à relancer vous-même dans votre terminal - la fenêtre où l'on tape des commandes, hors de la conversation avec l'assistant. Là, il nomme le geste et la base visée, et attend votre « oui ». L'assistant ne peut pas le donner par mégarde à votre place : il lui faudrait un contournement délibéré, que ses règles lui interdisent.

Ce que ça change pour vous

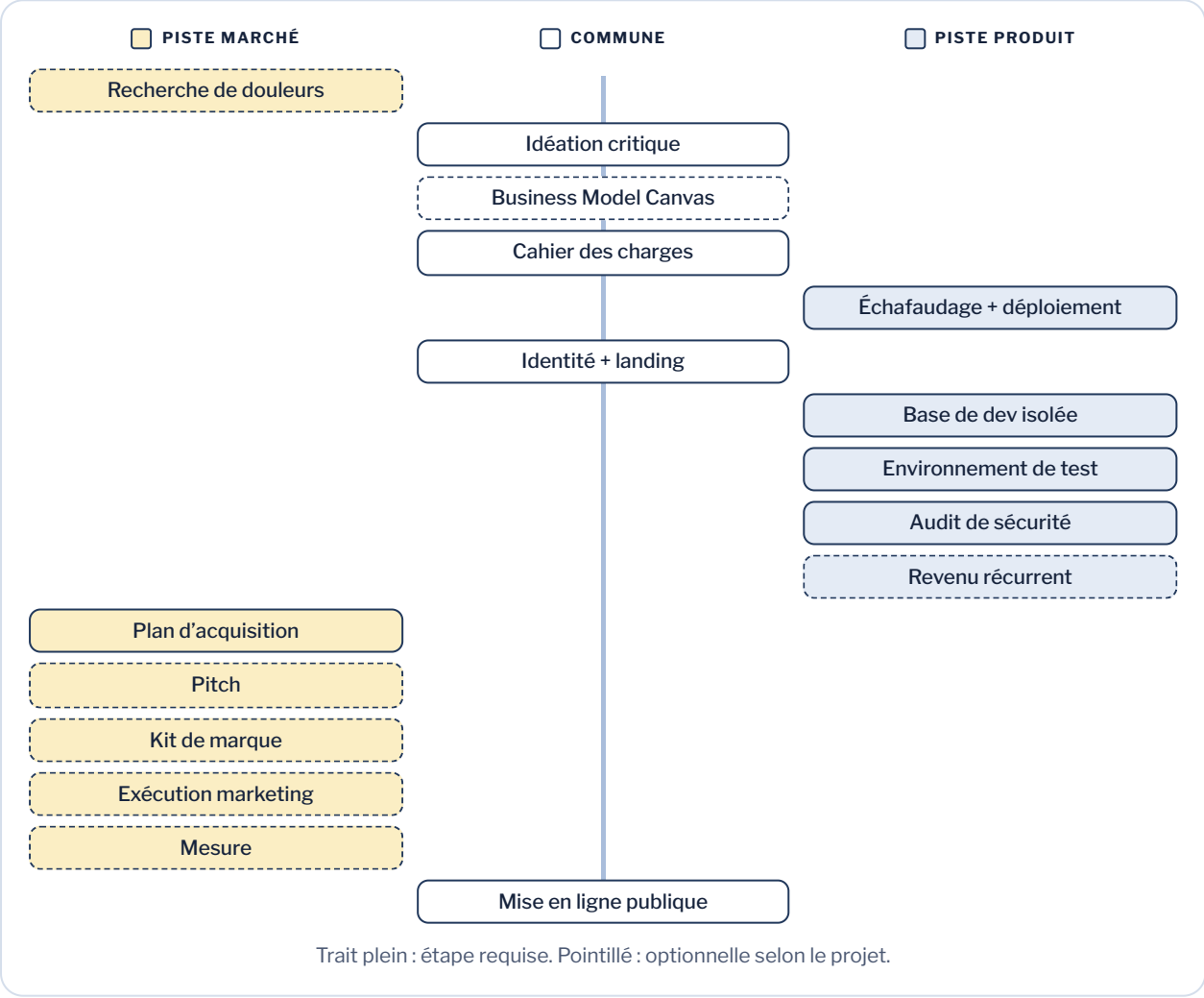
Vous ne devenez pas développeur. Vous devenez quelqu'un qui **livre**. La différence entre une idée et un produit n'est plus une compétence technique à acquérir pendant deux ans : c'est un parcours à suivre, et le chapitre suivant le décrit en entier.

CHAPITRE 02

Le parcours en entier

C’est le chapitre le plus important du livre. Tous les autres décrivent une étape ; celui-ci montre pourquoi elles s’enchaînent dans cet ordre.

La carte



Trait plein : étape requise. Pointillé : optionnelle selon le projet.

Les 16 étapes, et la raison de chacune

Cette liste est **lue dans le harnais** à la fabrication du livre, pas recopiée. Si une étape est ajoutée demain, elle apparaît ici sans que personne ne réécrive ce chapitre.

	COMMANDE	ÉTAPE	POURQUOI	PISTE	
01	/pain-point-finder	**Recherche de douleurs**	partir de douleurs documentées et sourcées, pas d'une intuition	marché	optionnelle
02	/ideate	**Idéation critique**	valider le problème, la cible et le go/no-go avant tout code	commune	**requise**
03	/bmc	**Business Model Canvas**	poser le modèle économique sur une page	commune	optionnelle
04	/spec	**Cahier des charges**	le cahier des charges détaillé que /bootstrap consomme	commune	**requise**
05	/bootstrap	**Échafaudage + déploiement**	créer le projet, le déployer en ligne et valider l'infrastructure	produit	**requise**
06	/design	**Identité + landing**	donner une vraie identité et une landing qui convertit, sortir du template	commune	**requise**
07	/db-isolate	**Base de dev isolée**	travailler sur une branche de base séparée, jamais sur la prod	produit	**requise**
08	/add-staging	**Environnement de test**	une URL de test stable pour éprouver le produit avant tout vrai utilisateur	produit	**requise**
09	/security	**Audit de sécurité**	auditer avant d'ouvrir au public (verdict PASS requis pour la prod)	produit	**requise**
10	/billing	**Revenu récurrent**	faire rentrer les renouvellements sans les relancer à la main	produit	optionnelle
11	/gtm	**Plan d'acquisition**	comment trouver et convertir les 100 premiers clients, chiffré	marché	**requise**
12	/pitch	**Pitch**	un pitch prêt pour un client, un partenaire ou un financeur	marché	optionnelle
13	/brand-kit	**Kit de marque**	l'identité publique complète (logo, visuels réseaux, favicon), dérivée des campagnes du plan d'acquisition	marché	optionnelle
14	/promote	**Exécution marketing**	exécuter concrètement l'acquisition sur les canaux choisis	marché	optionnelle
15	/metrics	**Mesure**	mesurer ce qui compte pour décider de la suite	marché	optionnelle
16	/launch	**Mise en ligne publique**	la checklist finale de mise en ligne, la livraison de valeur	commune	**requise**

Deux pistes, pas une file d'attente

Vous avez remarqué la colonne « piste ». Le parcours n'est pas une file unique : il a deux voies qui avancent en parallèle.

La piste **produit** construit la chose. La piste **marché** trouve les gens qui la paieront. Les étapes **communes** servent les deux.

Pourquoi cette séparation existe : une session consacrée au marketing à qui on répond « va construire un environnement de test » reçoit un conseil qui n'est pas faux, mais **hors sujet** - et une boussole hors sujet se fait ignorer, puis désinstaller. Le harnais regarde donc sur quelle piste vous travaillez avant de recommander quoi que ce soit.

Requis, optionnel, et pourquoi la distinction compte

Certaines étapes sont **requises** : sans elles, ce qui suit ne tient pas. D'autres sont optionnelles selon votre projet - un site vitrine sans comptes n'a pas besoin de tout ce qu'une application avec abonnements exige.

Le harnais ne vous bloque jamais sur une étape optionnelle. Il vous dit ce qu'elle apporterait, et vous décidez. Sur une étape requise, il insiste - et sur l'audit de sécurité avant mise en ligne, il refuse de passer outre. Cette étape-là n'est d'ailleurs cochée que sur un audit réussi : un audit fait ailleurs, ou resté sans verdict, ne compte pas.

La commande à retenir

Vous n'avez pas à mémoriser cette carte. Tapez `/next` : le harnais lit où vous en êtes - non pas ce que vous avez déclaré, mais **ce qui existe réellement** sur votre disque et en ligne - puis nomme l'étape suivante et sa raison, en une ligne.

Il vous dit aussi, en passant et seulement quand il y a quelque chose à dire, ce qui n'est pas une étape : les pages de votre cahier des charges qui manquent encore ; vos tests, s'ils échouent ou s'ils n'ont pas pu tourner ; les points du cahier des charges qui restent à préciser ; et la version de votre page d'accueil que vous avez retenue, avec les tailles d'écran sur lesquelles elle a été regardée.

Si vous préférez qu'il enchaîne tout seul aussi loin qu'il le peut, `/pilot` fait le vol complet : il vous pose toutes ses questions au départ, met de côté ce qui exige une décision humaine sans arrêter le reste, et vous présente à la fin ce qui est fait et ce qui vous attend, avec la raison de chaque point. Il ne construit jamais ce que vous avez écarté, et il ne choisit jamais à votre place entre plusieurs versions d'une page : ce choix-là vous attend à son retour.

CHAPITRE 03

Si votre projet n'est pas numérique

Ce chapitre existe parce que les deux précédents ont pu vous laisser croire que ce livre parle de logiciel. Il parle de méthode. La différence décide de qui peut s'en servir.

Une bonne part du parcours ne produit pas une ligne de code

Reprenez la carte du chapitre précédent et regardez ce que chaque étape fabrique. Une bonne part d'entre elles ne produit aucun code : elles produisent des documents, des chiffres et des décisions.

- `/pain-point-finder` : un dossier de douleurs sourcées - ce que les gens reprochent vraiment à ce qui existe, avec au moins deux sources citées et datées.
- `/ideate` : un verdict argumenté - go, no-go ou pivot. Le problème est-il réel ? Qui exactement souffre ? Que font ces gens aujourd'hui, et pourquoi feraient-ils autrement ?
- `/bmc` : un modèle économique sur une page - les neuf blocs, dans leur ordre de dépendance, et un compte de résultat mensuel dans votre monnaie.
- `/spec` : la description de ce que vous lancez. Elle est pensée pour une application ; pour un projet qui n'en est pas une, c'est sa discipline qui compte - écrire ce qui est dans la première version, et surtout ce qui attendra.
- `/gtm` : un plan d'acquisition chiffré - qui signe en premier, par quel canal, à quel coût, avec un plan daté pour les cent premiers clients.
- `/pitch` : un argumentaire, pour un client qui hésite, un partenaire, un financeur.
- `/brand-kit` : une identité visuelle - le logo, les couleurs, l'allure de chaque visuel.
- `/promote` : le matériel de campagne - les messages de prospection, le calendrier de contenu, les briefs de campagne avec ciblage et budget.

Un commerce, un atelier, une exploitation agricole, un service de transport, une auto-école : tous passent par ces étapes-là, et presque aucune ne demande de construire un site.

Deux exceptions à connaître, pour ne pas être surpris. `/brand-kit` part aujourd'hui de l'identité que `/design` pose dans un site : sans site, il vous enverra d'abord la créer. Et un visuel qui annonce un prix ou une date n'est déclaré fini qu'après relecture de la page en ligne qui vend l'offre : sans cette page, il reste un brouillon.

Ce que le harnais apporte vraiment

Ce n'est pas la construction. C'est une discipline de réflexion qui refuse de vous laisser sauter les questions qui coûtent cher.

Un cahier ne vous demande pas vos sources. Un tableur ne vous contredit pas. Un ami à qui vous exposez votre idée vous encourage, parce qu'il vous aime bien. Le harnais fait l'inverse : il exige des sources datées, il nomme la personne exacte qui paiera, il vous demande à quel prix le marché achète aujourd'hui et si vous l'avez vérifié ou supposé. Puis il énonce un verdict que vous n'aviez pas envie d'entendre.

Ce qui est structuré n'est pas la construction. C'est la réflexion.

Un exemple, du début à la fin

Prenons un projet qui ne touche à aucun écran : une savonnerie artisanale à Lomé. Voici ce que le parcours donne.

> `/pain-point-finder` - ce qu'on reproche aux savons vendus sur le marché : la peau qui tire, le parfum qui ne tient pas, le prix qui monte avec l'importation.
>
> `/ideate` - la contestation : combien de savonneries artisanales existent déjà à Lomé ? Qu'est-ce qui vous distingue vraiment ? À quel prix ce marché achète-t-il, et l'avez-vous vérifié ou supposé ?
>
> `/bmc` - le coût du beurre de karité et de la potasse, le prix de vente au détail et en gros, la marge par savon, le seuil de rentabilité mensuel en francs CFA.
>
> `/spec` - la gamme de départ : trois savons, pas douze. Et la liste de ce qui attendra la deuxième année.
>
> `/gtm` - qui achète en premier : les boutiques de quartier, les salons de coiffure, les revendeuses de marché. Le coût et l'effort de chaque canal, et lequel commencer.
>
> `/pitch` - l'argumentaire d'une minute pour la gérante d'une boutique qui hésite à prendre vos savons en dépôt.
>
> `/promote` - le calendrier de publications, les messages de prospection, et la discipline qui évite de passer pour un importun.

Au bout de ce chemin : pas de site, pas d'application, pas une ligne de code. Un projet documenté, chiffré et contesté avant la première dépense. C'est exactement ce que la plupart des projets ne font jamais, et c'est pourquoi la plupart des projets meurent de choses qu'on pouvait savoir d'avance.

Si un jour cette savonnerie veut vendre en ligne, les autres étapes l'attendent. Mais elles attendent : elles ne conditionnent rien. `/next`, qui suit le parcours complet, vous les proposera quand même ; sur un projet qui n'en a pas besoin, vous pouvez les laisser

de côté.

Ce que le harnais ne fera pas

Il ne visite pas votre local. Il ne goûte pas votre produit. Il ne négocie pas avec votre fournisseur, et il ne remplace pas une matinée passée au marché à regarder ce que les gens achètent vraiment. Il structure votre réflexion ; le terrain reste le vôtre, et il reste irremplaçable.

Une limite honnête, aussi : sur un marché très local, ses sources sont plus minces que sur un marché documenté. Quand la preuve est mince, il le dit au lieu de combler le trou, et c'est à vous d'aller chercher le reste.

> **LA RÈGLE DE CE CHAPITRE**
>
> **Un savon et une application passent par les mêmes étapes de réflexion, parce que ce qui se structure n'est pas ce qu'on construit : c'est la manière dont on décide de le construire.**

CHAPITRE 04

Avant de construire

L'erreur la plus chère n'est pas un bug. C'est six mois passés à construire quelque chose dont personne ne veut. Ce chapitre décrit les étapes qui coûtent quelques heures et en économisent des centaines.

Chercher la douleur avant d'avoir l'idée

Si vous n'avez pas encore d'idée, c'est une bonne position - meilleure que d'en avoir une à laquelle vous tenez déjà.

`/pain-point-finder` part d'un secteur, d'un pays ou d'un métier, et cherche des problèmes **documentés** : ce que les gens écrivent eux-mêmes sur les forums et dans les avis, la littérature académique, les rapports institutionnels, et ce que d'autres ont déjà tenté - y compris leurs échecs, qui sont la source la plus instructive.

Chaque douleur remontée porte **au moins deux sources indépendantes, citées et datées**. Jamais une référence inventée : quand la preuve est mince, le rapport le dit au lieu de combler le trou. Une opportunité fondée sur une source imaginaire est pire qu'aucune opportunité, parce qu'elle a l'air sérieuse.

Passer l'idée au feu

`/ideate` est un partenaire critique, pas un supporter. Il conteste : le problème est-il réel ou supposé ? Qui exactement souffre ? Que font ces gens aujourd'hui, et pourquoi votre solution serait meilleure que leur système D actuel ? Quel est le modèle économique défendable ? Qu'est-ce qui vous tuerait ?

Il ne valide jamais une idée par politesse. Un « non » argumenté est un succès de cette étape : il vous rend les mois que l'idée allait coûter. Le verdict est un go, un no-go ou un pivot, et il s'appuie sur ce qui a été trouvé, pas sur votre enthousiasme.

Quand le verdict est un go ou un pivot, il retient aussi quelques mots de votre métier - ceux qu'il faudra toujours employer, et ceux qu'il faudra écarter. Ils vont dans le glossaire du projet, que l'annexe décrit.

Écrire ce qu'on construit

`/spec` transforme le go en cahier des charges : les fonctionnalités, les écrans, ce qui est dans la première version et surtout **ce qui n'y est pas**.

La liste des exclusions est la plus utile du document. Une première version qui essaie de tout faire ne sort jamais. Le cahier des charges tient donc deux listes à part : ce qui reste **à préciser**, dans vos mots, et ce que vous avez mis **hors périmètre**, avec la raison. Ce n'est pas une précaution de forme : le harnais ne construit jamais ce que vous avez écarté, et il vous repose les points restés à préciser avant de les construire, plutôt que de les deviner. Quand la tentation d'ajouter reviendra - elle reviendra - c'est cette liste qui vous défendra.

Si vous avez du mal à choisir entre plusieurs allures pour votre produit, `/spec` peut vous proposer une maquette : un seul fichier, qui s'ouvre d'un double clic et montre trois versions très différentes de votre page d'accueil. Vous choisissez en regardant, et votre choix est noté dans le cahier des charges. C'est une proposition, jamais une obligation.

Ce document n'est pas de la paperasse : `/bootstrap` le lit et construit à partir de lui. Ce que vous écrivez ici devient le projet.

Le modèle économique, si l'argent est en jeu

`/bmc` déroule les neuf blocs du Business Model Canvas - une grille qui présente un projet d'entreprise sur une seule page - dans leur ordre de dépendance, en partant de ce que l'idéation a validé. Il conteste les blocs faibles - une source de revenu sans segment de clientèle en face, une activité clé que personne ne porte - et esquisse un compte de résultat mensuel dans votre monnaie.

La manière dont on vous interroge

`/ideate`, `/bmc` et `/spec` vous posent leurs questions de la même façon : une phase à la fois, une à trois questions par tour, et une réponse recommandée chaque fois que la question est un choix, avec ce que chaque option coûte. Rien n'est écrit avant que vous ayez confirmé que la machine vous a bien compris. `/pain-point-finder`, lui, vous pose quelques questions de cadrage, une par tour, puis part chercher.

> **LA RÈGLE DE CE CHAPITRE**
>
> **Ces étapes produisent des documents, pas du code. C'est exactement pour ça qu'elles sont rentables : un document se jette sans regret, du code qu'on a payé, non. La maquette que `/spec` peut vous montrer ne fait pas exception : elle est faite pour être regardée, puis jetée.**

CHAPITRE 05

Ce qui tourne sous le capot

Vous n'avez pas besoin de savoir programmer. Mais savoir **quelles pièces composent votre produit** vous sert trois fois : pour comprendre ce que vous payez, pour parler à un prestataire si un jour vous en prenez un, et pour ne pas être prisonnier d'un outil.

Ce chapitre est le seul du livre à nommer des technologies. Vous pouvez le sauter en première lecture.

Les pièces, et à quoi chacune sert

Le site lui-même est construit avec Next.js, le standard actuel du web. Il produit des pages rapides et bien référencées, et fonctionne aussi bien sur téléphone que sur ordinateur - le harnais l'exige à la même sévérité sur les trois tailles d'écran, ce qui n'est pas un détail dans un marché où la majorité des visiteurs arrivent par mobile.

La base de données est PostgreSQL, hébergée chez Neon. C'est là que vivent vos comptes utilisateurs, vos commandes, votre contenu. Elle est facturée à l'usage, avec un palier gratuit qui suffit largement à démarrer.

L'hébergement est Vercel : votre site y est publié, mis à jour à chaque changement, et servi depuis une région choisie selon votre marché. Cette dernière précision vaut de l'argent - une base à Francfort et un serveur à Paris se parlent plusieurs fois par page affichée, et le harnais les apparie pour éviter cette latence payée en silence.

Les paiements passent par Stripe ou par un fournisseur adapté à votre marché - en zone UEMOA, FedaPay, qui accepte le mobile money que vos clients utilisent réellement.

Les emails partent par Resend ou Brevo : confirmations de compte, réinitialisations de mot de passe, notifications.

L'intelligence dans votre produit, si vous en mettez - un assistant qui répond, une étape qui trie ou résume, un agent - passe par OpenRouter, un intermédiaire qui donne accès à de nombreux modèles. Par défaut, le harnais y choisit des modèles ouverts, bien moins chers : ce coût-là revient chaque mois, et c'est votre produit qui le paie. Claude reste proposé comme option haut de gamme, avec son prix du jour, et vous choisissez tâche par tâche.

Ne confondez pas les deux plans. Pour **construire** votre produit, le harnais recommande, aux étapes qui comptent, le modèle le plus capable, parce qu'une erreur de construction coûte cher. Ce que votre produit **consomme** ensuite est une autre dépense, et elle se décide à part.

Ce que ça coûte pour démarrer

Presque tout ce qui précède a un palier gratuit, et le harnais est conçu pour y rester le plus longtemps possible. `/quotas` vous montre où vous en êtes sur chaque service, et vous alerte **avant** la falaise plutôt qu'après. Quand le mur approche, `/optimize` cherche dans votre code ce qui le consomme : le chapitre 7 le décrit.

La première dépense n'est pas dans cette liste. Pour exécuter ces commandes, il faut un accès à l'assistant qui les exécute, et ce livre en décrit deux. **Antigravity**, l'hôte de Google : son palier gratuit s'utilise tel quel, sans abonnement, et il comporte une limite hebdomadaire que Google ne publie pas. **Claude Code**, l'hôte d'Anthropic : par abonnement, par un compte facturé à l'usage - souvent la voie la moins chère pour commencer - ou, dans le terminal et dans VS Code seulement, par un fournisseur tiers ; l'application de bureau, elle, exige un abonnement payant. Le harnais tourne sur les deux, mêmes commandes, mêmes projets. Si la dépense compte, commencez par la porte gratuite. Ces conditions ont été relevées en août 2026 : `/about` vous dit ce qu'il faut posséder aujourd'hui, avec la source et la date.

Quand vous mettez de l'intelligence dans votre produit avec `/add-ai`, vous voyez un devis avant la première ligne de code, et votre projet reçoit sa propre clé. Une clé, c'est le mot de passe qu'un service remet à votre projet pour qu'il s'en serve en votre nom : qui la détient peut agir et dépenser à votre place, d'où les précautions que vous croiserez dans ce livre. Celle-ci porte un plafond de dépense tenu par le fournisseur lui-même - le harnais le pose pour vous quand il le peut, et vous demande de le poser sinon : une boucle qui s'emballe bute sur ce mur-là, pas sur une promesse.

> ⚠ Un point qui n'est pas technique mais juridique : un plan gratuit n'autorise pas forcément un usage commercial. `/commercial-check` lit les conditions de chaque service que votre projet utilise et vous dit ce qu'il faut régler avant de vendre. Il échoue fermé : une condition non vérifiée n'est jamais présentée comme un feu vert. Et si vendre vous oblige à prendre un abonnement payant chez l'hébergeur, il vous présente les alternatives, chacune avec ce qui la bloque - une clause qui interdit la vente, une coupure sans préavis, un crédit gratuit qui ne tient pas le mois - sans jamais prétendre qu'une offre gratuite règle la question.

Le mot qui compte : vous n'êtes pas enfermé

Tout ce que le harnais construit vous appartient. Le code est dans votre dépôt, la base est sur votre compte, le nom de domaine est à votre nom. Si vous décidez un jour de continuer sans le harnais, ou avec quelqu'un d'autre, **rien ne vous retient** - c'est du code standard que n'importe quel développeur du métier sait reprendre.

CHAPITRE 06

Le démarrage du projet

`/bootstrap` est la commande qui transforme un dossier vide en projet en ligne. En une séance, vous passez d'une idée écrite à une adresse que vous pouvez ouvrir dans un navigateur et montrer à quelqu'un.

Ce qu'il vous demande, et pourquoi

Il pose peu de questions, et chacune décide de quelque chose qu'on ne peut pas deviner.

Ce que vous construisez - si vous avez un cahier des charges, il le cherche et le lit au lieu de vous refaire raconter le projet. Ce que vous y avez mis hors périmètre n'est pas construit, et ce qui reste à préciser vous est demandé avant d'être construit.

Site ou application web, ou application mobile - posée avant toute question technique, parce que toute la suite en dépend. Le harnais ne devine pas cette réponse : deviner mal produirait un site pour quelqu'un qui voulait une application, chaque étape annonçant un succès.

Pour quel marché - le pays et la monnaie décident du fournisseur de paiement proposé, de la région d'hébergement, des canaux marketing réalistes plus tard. Un projet togolais et un projet français ne se construisent pas pareil, et le deviner mal est pire que demander.

Qui publie le projet - la raison sociale, l'adresse du siège, le responsable de publication. Ces informations partent dans vos mentions légales, un document opposable. Si la machine a déjà servi à publier pour une entreprise, elle vous la **propose** - et vous la confirmez à voix haute, même s'il n'y en a qu'une.

Cette dernière insistance a une raison : redemander huit champs coûte deux minutes, tandis que publier le numéro d'immatriculation d'un client sur le site d'un autre est une faute dans un document légal, et une faute **invisible** - personne ne relit ses mentions légales après la mise en ligne.

Ce qu'il fait sans le demander

Le dépôt de code créé et privé par défaut. Le socle du site posé. La base de données provisionnée dans la bonne région. Le site déployé et vérifié - non pas « la commande a répondu », mais **le déploiement est allé au bout et la page répond**. Les premiers tests automatiques écrits sur ce qu'il vient de poser, et l'outil qui les relit sans confondre « une

faute trouvée » et « l'outil n'a pas démarré ». Les pages légales écrites, avec la loi de votre pays et non celle d'un autre continent. Le plan du site, le référencement de base, le fichier qui dit à Google quoi indexer.

Et un point qu'on remarque seulement quand il manque : **votre projet porte sa licence et son attribution dès le premier commit**. Si vous vendez ce projet à un client, il est en règle le jour de la livraison.

Un autre, qu'on ne remarque jamais : quand il doit relire vos réglages de production, il ne les copie jamais dans votre projet, d'où ils pourraient partir par erreur avec votre code.

À la fin

Vous avez une adresse. Elle affiche une page d'accueil sobre, elle est en ligne, elle est à vous. Ce n'est pas encore votre produit - c'est le socle sur lequel les chapitres suivants ajoutent ce qui le rend utile.

Le harnais vous annonce alors l'étape suivante et sa raison. Suivez-la.

Ce que la question mobile décide

Une précision, parce qu'elle décide de votre architecture. Le harnais construit des sites et des applications web complets, et c'est là que presque toutes ses commandes tournent. Sur une application mobile, il en fait déjà beaucoup - l'application, son serveur, les comptes de vos utilisateurs, la base de données et son isolement, les tests - mais pas encore tout. Il construit aujourd'hui une application Android installable ; l'iPhone n'a pas encore été éprouvé, et sa compilation exige un compte développeur Apple payant. `/about` vous dit à tout moment, commande par commande, ce qui tourne sur mobile et ce qui refuse encore : c'est lui qui fait foi, pas cette page.

Du côté de l'argent, la vraie question n'est pas « mobile ou web » : c'est **ce que vous vendez**.

Les boutiques d'applications distinguent deux choses. Ce qui est livré dans le monde réel - une course, une livraison, un repas, une marchandise, une prestation - ne relève pas de leur système d'achat intégré : votre application encaisse librement, par mobile money ou par carte. C'est ainsi que Gozem vous fait payer une course. Ce qui est consommé **dans** l'application - un abonnement, des crédits, du contenu premium - doit au contraire passer par leur système d'achat intégré, lequel n'accepte pas le mobile money. Là, la porte est fermée : pour notre marché, ce n'est pas une commission, c'est un moyen de paiement qui n'existe pas chez eux.

Ce que le harnais sait faire aujourd'hui :

- **FedaPay encaisse dans une application mobile.** La page de paiement s'ouvre dans le navigateur du téléphone - le mobile money doit pouvoir passer la main à l'application de votre portefeuille - et l'application attend que le serveur ait confirmé le paiement avant

de l'annoncer. Ce chemin a été éprouvé de bout en bout, par un paiement d'essai depuis une application Android installée.

- **Stripe sur mobile est écrit, mais n'a encore jamais été éprouvé, même par un paiement d'essai.** La commande vous le dit avant de commencer, et ne continue qu'avec votre accord. Avant d'écrire le paiement dans l'application, elle vous demande ce que l'application vend, et refuse le contenu numérique consommé dans l'application, en expliquant pourquoi : le générer ferait rejeter votre application à la revue, des semaines plus tard.
- **Pour du contenu numérique consommé dans l'application, soyez prudent.** Le harnais part par défaut du principe que l'on s'abonne sur le web et que l'application ne fait que consommer un droit déjà payé. Mais les boutiques encadrent de près cette façon de faire, et `/add-fedapay` ne vous pose pas encore la question que pose `/add-stripe` : c'est à vous de vous la poser, avant de construire.

Cette question décide si votre modèle tient sur mobile. Nous n'annonçons une capacité qu'après l'avoir vue fonctionner pour de vrai : c'est cette règle qui vous protège, et elle vaut aussi quand elle nous ralentit.

CHAPITRE 07

Les outils du quotidien

Une fois le projet né, quelques commandes reviennent presque chaque séance. Les connaître change le rythme de travail.

Savoir où vous en êtes

`/next` répond à « et maintenant ? ». Il ne se fie pas à ce que le projet **déclare** avoir fait : il regarde ce qui existe réellement - les documents sur le disque, la base, la branche de test, le déploiement, les tests - puis recommande une étape avec sa raison.

C'est la commande à retenir si vous n'en retenez qu'une.

Donner un visage au produit

`/design` donne à votre produit son identité : couleurs, typographie, ton. Il ne vous demande pas de la choisir sur des adjectifs ou des codes couleur : il vous montre trois directions visuelles, chacune avec son ambiance, sa paire de polices et son élément signature, puis trois versions de votre page d'accueil, construites dans votre projet et affichées sur votre ordinateur, sans rien publier. Vous choisissez **en voyant**. Sous Claude Code, chaque version est d'abord regardée sur téléphone, sur tablette et sur ordinateur ; ailleurs, le harnais vous dit qu'elle ne l'a pas été. Il écrit ensuite ce choix dans le projet et habille les boutons et les champs à l'identique, ce qui évite que chaque nouvelle page réinvente une apparence. Au passage, il garde un contour visible pour qui navigue au clavier, et respecte les personnes qui ont réduit les animations sur leur appareil.

`/brand-kit` va plus loin, dans la même ambiance : logo, couvertures pour les réseaux, gabarits de publication - et surtout les **consignes de production** de chaque visuel, prêtes à être exécutées.

`/generate` produit les fichiers eux-mêmes. C'est la famille de commandes qui dépense de l'argent réel au moment même où vous la lancez : elle annonce le prix avant chaque production, ne dépense rien sans confirmation explicite, télécharge immédiatement, et écrit à côté de chaque fichier un journal disant quel modèle, quel prix, quelle consigne. Une dépense sans trace est une dépense qu'on refait par accident.

`/montage` assemble et finit - recadrage, texte incrusté, assemblage de plans vidéo. Cette moitié-là est **gratuite** : c'est du traitement local. Et avant de déclarer fini un visuel qui annonce une date, un prix, un lieu ou un contact, il relit ce fait sur votre page en ligne : une

affiche qui contredit votre site coûte plus cher qu'une affiche en retard.

Vérifier avant de montrer

`/security` audite : secrets exposés, routes non protégées, validation des entrées, dépendances vulnérables, en-têtes. Son verdict est enregistré, et la mise en ligne le consulte. Il nomme chaque accès de votre application resté sans décision - public ou réservé - et vous envoie vers `/add-api-guard` pour le fermer. Il fait aussi relire le code par les outils de vérification automatique, et sait distinguer un outil qui a trouvé une faute d'un outil qui n'a pas démarré : les deux répondent de la même façon, un seul a vérifié quelque chose. Si votre projet n'a pas encore de tests, il les installe avant de noter.

`/seo` travaille le référencement, `/seo-perf` mesure la vitesse réelle chez Google plutôt que de la supposer, `/eco-audit` calcule l'empreinte de vos pages. `/seo` veille aussi à l'image qui accompagne vos liens partagés : il la range de sorte que chaque nouvelle image porte une nouvelle adresse, ce qui permet à WhatsApp et Facebook d'afficher la nouvelle plutôt que l'ancienne gardée en mémoire, parfois après un délai.

`/rgpd-audit` tient à jour le registre de vos sous-traitants et la page de confidentialité. Cette page est un document opposable, et le harnais y met la loi de votre pays. Il y nomme les vrais destinataires de vos données - y compris l'intermédiaire par lequel passe l'intelligence de votre produit - et n'invente aucune durée de conservation qu'un fournisseur ne publie pas.

Surveiller ce que ça coûte

`/quotas` fait le tour de vos services et dit où vous en êtes sur chaque palier gratuit - avec les falaises nommées, celles dont le franchissement **suspend** le service plutôt que de le facturer.

`/optimize` prend le relais quand une falaise approche : il cherche dans votre code ce qui la consomme, dit quel palier chaque cause brûle, commence par l'urgent, et ne touche au code qu'avec votre accord. `/quotas` mesure ; `/optimize` explique et répare.

`/save-project` prend un instantané : base, fichiers, configuration. `/clean` range - et relit d'abord ce que votre projet a gardé exprès, pour ne pas vous proposer de supprimer un module conservé à dessein.

Quand un secret a fuité

`/rotate-secret` remplace une clé compromise sur votre ordinateur et sur votre site, et sur les serveurs de vos agents pour les clés qu'il sait y chercher. Il ne redéploie chaque service qu'après avoir vérifié qu'il a bien reçu la nouvelle valeur. Remplacer un secret de production vous est toujours demandé, quel que soit votre niveau d'autonomie.

Quand vous ne savez plus

`/about` dit quelle version tourne, ce qu'elle sait faire sur un site et sur une application mobile, et ce qu'il faut posséder pour s'en servir. Il vous dit aussi une chose utile sur une installation neuve : l'assistant ne trouve pas toujours seul la bonne commande quand vous décrivez votre besoin sans la nommer. Rien n'est cassé : tapez la commande, ou tapez `/next`, qui vous la nommera.

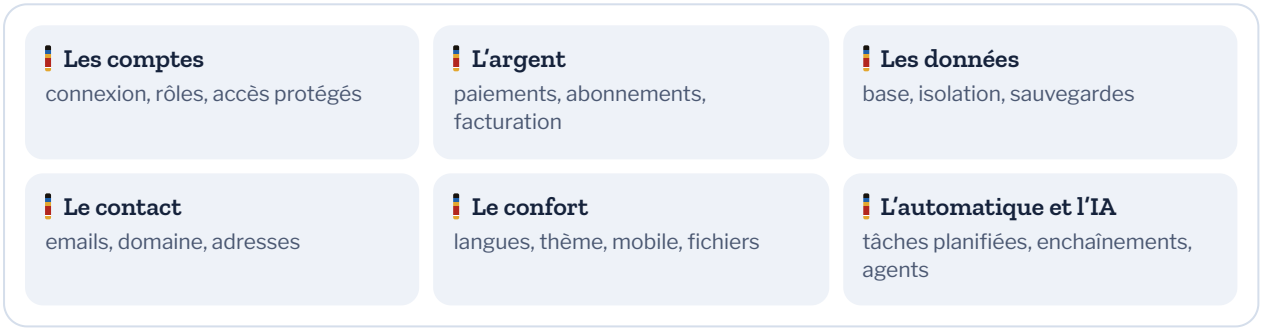
`/explain` redit simplement ce que vous n'avez pas compris : un mot, une phrase, un écran, une erreur, une décision qu'on vient de vous proposer. `/diagnose` prend le relais quand quelque chose ne marche pas. `/prof` enseigne - c'est le cours dont ce livre est tiré. Et `/next`, toujours.

CHAPITRE 08

Ajouter une capacité

Votre site existe. Il lui manque des comptes utilisateurs, des paiements, un formulaire de contact. Chaque capacité s'ajoute par une commande qui écrit le code, pose la configuration, et met à jour la documentation du projet.

Les grandes familles



Les comptes. `/add-auth` installe l'authentification. Par défaut il crée **des comptes utilisateurs ET un administrateur** - parce qu'un produit avec des utilisateurs et sans personne pour les administrer est un produit qu'on découvre incomplet des semaines plus tard, une fois l'application bien avancée. `/add-google-auth` et `/add-github-auth` ajoutent la connexion par un compte existant ; `/add-2fa` la double authentification ; `/add-role` les rôles.

`/add-api-guard` ferme ensuite les accès par défaut : une nouvelle adresse technique de votre application est réservée aux personnes connectées tant que vous n'avez pas écrit pourquoi elle serait publique, et il vous fait classer une à une celles qui existent déjà. Il installe aussi la protection qui empêche un client connecté de lire les données d'un autre en changeant un numéro dans l'adresse. Elle protège chaque accès qui l'emploie, et l'audit de sécurité vous signale ceux qui ne l'emploient pas encore.

L'argent. `/add-stripe` pour les cartes internationales, `/add-fedapay` pour le mobile money d'Afrique de l'Ouest. `/billing` suit les abonnements. Avec FedaPay, dans un projet qui a sa base de données, chaque paiement a sa ligne : il passe d'« en attente » à « approuvé » sur la confirmation signée que FedaPay envoie, et votre page de retour lit ce statut au lieu de croire le lien par lequel le client revient. Le prix, lui, est retrouvé par le serveur à partir du produit : un client ne peut pas payer un article cher au prix d'un article bon marché.

La bascule en mode réel - celle où de vrais euros ou francs circulent - exige toujours votre accord explicite, et aucun niveau d'autonomie ne vous en dispense. Pour FedaPay, dans un projet qui a sa base de données, elle exige en plus la preuve qu'un paiement d'essai a réellement été approuvé ; sans base, cette preuve est impossible, le harnais vous le dit et vous demande une confirmation explicite. Les clés réelles ne vont qu'en production, si bien que votre site de test ne peut pas encaisser de vraies cartes, et une clé mal collée - un espace, un caractère invisible, le mauvais mode - est arrêtée avant de partir, avec un message qui dit quoi corriger. Si vous voulez revenir en arrière, Stripe a une étape guidée qui remet réellement votre site public en mode test, après un redéploiement qu'elle vous propose.

Les données. `/add-db` installe la couche de données. `/db-isolate` sépare votre base de travail de la base réelle : c'est l'étape du chapitre suivant, et la sauter est la faute la plus coûteuse du parcours. `/add-backup-db` programme les sauvegardes. Et même sur vos bases de travail et de test, une instruction qui efface une table, ou toutes ses lignes, est d'abord arrêtée : l'assistant doit vous demander votre accord avant de la relancer. C'est un garde-fou contre l'étourderie, pas un mur : seule la base de production refuse elle-même d'écrire.

Le contact et la présence. `/add-email` pour les envois, `/add-domain` pour votre propre nom de domaine, `/new-email-address` pour une adresse à ce nom. La clé d'accès de votre fournisseur de domaine ne passe jamais par la conversation, où elle resterait pour toujours : vous la collez dans une petite fenêtre à part, où elle reste masquée, et elle est rangée dans un coffre-fort de mots de passe (Bitwarden), que le harnais vous aide à installer la première fois qu'il en a besoin. Et avant de créer une adresse, le harnais vérifie la messagerie déjà en place sur le domaine, pour ne pas couper votre courrier.

Le confort. `/add-i18n` pour plusieurs langues, `/add-dark-mode`, `/add-pwa` pour l'installation sur téléphone, `/add-map`, `/add-storage` pour les fichiers et images, `/add-analytics` avec son bandeau de consentement.

L'automatique et l'intelligence. `/add-ai` pour une étape intelligente déclenchée par une personne - un assistant qui répond, un tri, un résumé : une question, une réponse, sans boucle. `/add-cron` pour les tâches à heure fixe, fermées à qui n'a pas leur secret. `/add-workflow` pour une suite d'étapes déclenchée par un événement, `/add-agent` pour un agent qui décide seul et tourne en continu. Si vous ne savez pas laquelle choisir, `/add-automation` vous écoute décrire le besoin et vous envoie vers la bonne.

La différence entre un enchaînement et un agent n'est pas l'intelligence, c'est la **boucle** : un enchaînement fini se termine, un agent tourne.

Ce qui pense dans votre produit part par défaut sur des modèles ouverts, bien moins chers ; le chapitre « Ce qui tourne sous le capot » dit pourquoi. Un agent, lui, naît fermé : il n'envoie d'email qu'aux adresses que vous avez déclarées, n'envoie de données vers aucun

site non déclaré, et ne lit votre base qu'en lecture seule, imposée par le serveur. Une instruction cachée dans une page ou un email qu'il lit ne peut donc pas lui faire expédier vos données en masse à un inconnu. Il garde pourtant le droit de consulter n'importe quelle page, et une consultation peut emporter quelques mots dans son adresse : ne donnez à un agent que les outils dont sa mission a besoin. Son budget, par jour et par mois, est vérifié avant chaque exécution et après chaque tour : une boucle qui s'emballe s'arrête dès qu'elle le franchit, au lieu de tourner toute la nuit.

Ce qu'elles font toutes, sans le dire

Chacune vérifie que ce dont elle dépend est déjà là, refuse proprement en le nommant sinon, écrit sa configuration aux bons endroits, met à jour la documentation du projet, et déclare ses nouveaux sous-traitants au registre de confidentialité quand elle en introduit.

Ce dernier point est celui qu'on oublierait toujours à la main : ajouter une carte interactive envoie l'adresse IP de chaque visiteur chez un tiers, et votre page de confidentialité doit le dire. Après avoir ajouté de l'intelligence à votre produit, relancez tout de même `/rgpd-audit` : c'est lui qui lit dans le code à qui partent réellement vos données.

Plusieurs posent aussi leurs propres tests - les mots de passe, les accès protégés, la confirmation de paiement, les tâches planifiées - que la mise en ligne relira. Et si ce qu'une commande s'apprête à faire n'a encore jamais tourné pour de vrai sur votre type de projet, elle vous le dit avant, pas après, et attend votre accord.

La règle : une capacité à la fois

Ajoutez, regardez le résultat, validez. Trois capacités ajoutées d'un coup produisent un projet dont vous ne savez plus laquelle a cassé quoi.

CHAPITRE 09

Mettre en ligne, en trois temps

Voici le chapitre où l'on perd de l'argent quand on va trop vite.

Trois environnements, toujours



Le palier de test passe avant le public : le premier essai de paiement vous revient, pas à un client.

LOCAL - sur votre machine. Vous construisez, vous cassez, vous recommencez. La base est une copie de travail, jamais celle de vos clients. C'est `/db-isolate` qui l'établit, et c'est une étape requise.

TEST - une copie privée en ligne, sur une adresse stable. Base de test, paiements en mode bac à sable. C'est **ici que vous essayez pour de vrai** : créer un compte, faire un achat de bout en bout, comme le ferait un client. C'est `/add-staging` qui la pose.

PRODUCTION - le site public. On n'y va qu'après votre validation explicite du palier de test.

Pourquoi trois et pas deux

Parce que le premier test de paiement doit être fait par vous avec de l'argent factice, et non par votre premier client avec le sien.

Et parce qu'une base de test vous laisse essayer les cas laids - créer, supprimer, refaire - sans risquer les données de gens réels. Un test sur la base de production n'est pas un test, c'est une manipulation en direct.

Ce que le harnais vérifie à votre place

Sur le palier de test comme en production, il ne se contente pas de lancer le déploiement : il **vérifie que la construction a réussi**, sur la bonne branche, et pour la bonne version - celle de votre dernier envoi, pas la précédente. Cette précision a une histoire - demander « le dernier déploiement de test » sur un projet à plusieurs branches vivantes répond sur le

déploiement de quelqu'un d'autre, et ce verdict ressemblerait à un succès tout en ne prouvant rien. Et si le réseau coupe pendant la vérification, il vous le dit comme tel, au lieu de vous faire chercher une panne.

Il vérifie aussi qu'une adresse de test n'est pas derrière un mur de connexion : une copie privée que vous seul pouvez ouvrir ne sert pas à faire tester quelqu'un.

Quand il envoie un réglage vers l'hébergeur, il vérifie avant de supprimer quoi que ce soit, et vous dit exactement quels environnements l'ont reçu : la production ne peut pas perdre sa base parce qu'on préparait le palier de test.

Et avant le GO de `/launch`, il relit les tests du projet : un test qui échoue bloque, et un outil de test qui n'a pas pu tourner bloque aussi - rien n'a vérifié le code, ce n'est pas « aucune faute ».

La séquence, en clair

1. Travailler et tester en **local**.
2. Enregistrer le travail et vous le montrer, pour validation.
3. Pousser sur la branche de **test**, et vous inviter à l'essayer en conditions réelles.
4. **Seulement après votre accord explicite**, publier en production.

Jamais de raccourci vers l'étape 4. Vous pouvez configurer le harnais pour qu'il soit plus autonome sur les étapes 2 et 3 - l'envoi vers la branche de test peut alors partir sans vous demander. L'étape 4 reste la vôtre, et aucun réglage ne vous en dispense.

Au niveau le plus autonome, quelques déploiements techniques peuvent partir seuls : la mise en ligne d'une tâche planifiée ou d'un petit service d'automatisation, la remise en route d'un service après un changement de domaine, le redémarrage du site après le remplacement d'une clé. Ils ne partent que si la vérification du code passe et si l'audit de sécurité porte sur le code tel qu'il est aujourd'hui : un audit réussi avant les derniers changements ne compte plus, même s'il date d'hier. L'autonomie ne s'achète pas d'un simple réglage.

La dernière vérification

`/launch` est la vérification d'avant-ouverture : l'audit de sécurité est-il passé, les valeurs d'exemple ont-elles été remplacées, les conditions commerciales des services que vous utilisez sont-elles réglées, le code et les tests ont-ils réellement été vérifiés. Il bloque sur ce qui doit bloquer.

Il vous montre aussi ce que votre page annonce quand on partage un lien vers votre site sur WhatsApp ou Facebook - le titre, la description, l'image. Et chaque annonce qui porte une date, un prix ou un lieu fait relire ce fait sur votre page en ligne le jour où elle part.

Le reste - le nom de domaine, les emails à ce nom - se pose avant ou après, mais la vérification, elle, passe avant. `/add-domain` vous demande votre accord avant de confier la gestion de votre nom de domaine à un autre service - ce qu'on appelle déplacer ses serveurs de noms -, un geste qui ne se défait pas d'un clic ; et si vous changez un jour de domaine, il met à jour les paiements et les services de ce projet, jamais ceux de vos autres projets.

CHAPITRE 10

Trouver des clients

Un produit en ligne que personne ne connaît ne rapporte rien. Cette moitié du parcours est celle qu'on saute le plus souvent, et c'est celle qui décide si le reste aura servi.

Le plan, avant les publications

`/gtm` construit le plan d'acquisition de **votre** produit - pas du marketing générique. Il lit ce que l'idéation a validé et le marché déclaré du projet, puis travaille quatre choses :

Qui signe en premier. Pas « les PME » : la personne précise qui a le problème, celle qui paie, celle qui peut bloquer. Ces trois-là sont parfois trois personnes différentes, et le message n'est pas le même.

La proposition de valeur en une phrase, dans les mots du client - pas dans les vôtres.

Les canaux réalistes de votre marché, avec un coût et un effort par canal. Ce qui marche à Lomé n'est pas ce qui marche à Paris, et le harnais ne recommande pas un canal que vos clients n'utilisent pas.

Un plan daté pour les cent premiers clients, avec des objectifs hebdomadaires et un budget dans votre monnaie.

Puis le visage, puis les mots

L'ordre compte, et il a été appris à ses dépens : `/gtm` d'abord, `/pitch` ensuite, `/brand-kit` en dernier. Un kit visuel construit avant le plan produit des visuels pour des canaux que le plan écartera - c'est arrivé, et c'était payé à l'image.

`/pitch` écrit l'argumentaire : la page de vente, le message de prospection, la présentation courte. `/brand-kit` habille ensuite ce qui a été décidé.

Exécuter

`/promote` transforme le plan en matériel prêt à lancer : séquences de messages avec la discipline anti-spam intégrée, calendrier de contenu, briefs de campagne publicitaire avec ciblage et budget, plan de mesure.

> ⚠ Il **propose** les campagnes ; c'est vous qui les lancez. Toute dépense publicitaire passe par votre accord.

Ne jamais annoncer ce que la page ne dit plus

Une affiche, un message, un aperçu de lien partagé portent souvent un fait de votre offre : une date, une heure, une durée, un lieu, un prix, un contact. Si la page qui vend a changé entre-temps, l'annonce ment - et c'est la page qu'on croira, pas l'affiche.

Le harnais relit donc ces faits sur votre page réellement en ligne avant de déclarer un visuel fini, et `/promote` note, pour chaque publication datée, l'adresse de la page à relire le jour même. Un fait absent de la page n'arrête pas la fabrication du visuel ; il arrête sa remise, jusqu'à ce que vous ayez tranché.

Mesurer, et corriger

`/metrics` installe le suivi : combien de visiteurs, combien s'inscrivent, combien paient, et où ils abandonnent. Sans ces chiffres, améliorer un produit revient à deviner.

L'endroit où les gens abandonnent est presque toujours une surprise. C'est pour ça qu'on mesure au lieu de supposer.

> LA LEÇON DE CE CHAPITRE
>
> **La piste marché avance en parallèle de la construction, pas après. Le meilleur moment pour savoir qui achètera est avant d'avoir fini de construire - il est encore temps de changer ce qu'on construit.**

CHAPITRE 11

Bien décrire ce qu'on veut

C'est la compétence centrale de cette manière de travailler, et la seule que le harnais ne peut pas exercer à votre place.

Le principe : décrire le résultat, pas la méthode

« Fais une page où les clients voient leurs commandes passées, de la plus récente à la plus ancienne, avec le statut de chacune » est une bonne demande.

« Crée un composant React avec un tableau trié » en est une mauvaise - non parce qu'elle est technique, mais parce qu'elle **impose une solution** avant d'avoir décrit le besoin. Si le tableau n'était pas la bonne forme sur téléphone, vous venez d'interdire la bonne réponse.

Dites **quoi** et **pour qui**. Laissez le comment.

Les quatre questions qui rendent une demande exécutable

Avant de demander quelque chose, vérifiez que votre phrase répond à ceci :

Qui utilise cette fonctionnalité ? Un visiteur, un client connecté, un administrateur ? La réponse décide des protections.

Quoi doit-il pouvoir faire, concrètement ?

Quand cela peut-il mal se passer ? Une commande annulée, un paiement refusé, un champ vide. Les cas laids sont ceux qu'on oublie de décrire, donc ceux qui manquent au résultat.

Comment saurez-vous que c'est réussi ? Si vous ne pouvez pas décrire la vérification, la demande n'est pas encore assez précise.

Reconnaître un résultat, sans lire le code

Vous ne relisez pas le code. Vous vérifiez le **comportement**, et cela s'apprend vite :

Essayez le cas normal, puis **essayez de casser**. Champ vide, texte à la place d'un nombre, double clic sur le bouton d'achat. Regardez sur téléphone, pas seulement sur ordinateur - la majorité de vos visiteurs y sera.

Et si quelque chose vous paraît étrange, dites-le, même sans savoir l'expliquer. « Cette page met du temps à s'afficher » est un rapport parfaitement utile.

Ce qu'il ne faut pas faire

Ne validez pas ce que vous n'avez pas regardé. Un « ça a l'air bon » sur un écran jamais ouvert est la manière la plus sûre de découvrir un défaut chez un client.

Ne demandez pas trois choses dans une phrase. Vous ne saurez pas laquelle a mal tourné.

Ne dites pas « comme d'habitude ». Chaque conversation repart d'un contexte propre ; ce qui a été décidé se réécrit ou se range dans le projet. Le harnais y tient trois endroits pour ça : un manifeste pour les réglages - votre pays, votre monnaie, votre fournisseur de paiement - qu'on ne vous redemande plus ; un registre pour les décisions qu'on ne veut pas rejouer, que plusieurs commandes relisent avant de vous proposer le contraire ; et un glossaire pour vos mots, décrit en annexe.

Ne laissez pas un flou en l'état. Si vous ne savez pas encore trancher un point, dites-le : il s'écrit dans la liste de ce qui reste à préciser, et le harnais vous le reposera avant de le construire. Un point flou qu'on laisse deviner revient toujours sous une forme que vous n'aviez pas choisie.

Le raccourci

Si tout cela vous paraît beaucoup : `/spec` pose ces questions à votre place, dans l'ordre, et écrit le document. Vous n'avez qu'à répondre.

CHAPITRE 12

Travailler avec la machine

Quelques habitudes qui séparent une séance productive d'une séance frustrante.

Une chose à la fois

Demandez une modification, regardez-la, validez-la. Puis la suivante.

Trois demandes empilées produisent un résultat où vous ne savez plus quoi a cassé quoi, et revenir en arrière devient une négociation. Le harnais enregistre le travail par petits paliers pour cette raison exacte : un palier validé est un point de retour.

Dire ce qui ne va pas, sans savoir pourquoi

Vous n'avez pas à diagnostiquer. « Le bouton ne fait rien quand je clique deux fois vite » est un excellent rapport - meilleur qu'une hypothèse sur la cause, qui enverrait chercher au mauvais endroit.

Décrivez **ce que vous avez fait, ce que vous attendiez, ce qui est arrivé**. Ces trois phrases suffisent presque toujours.

Et c'est exactement ce que `/diagnose` vous demande. Donnez-lui ces trois phrases : il vérifie lui-même le déploiement, les journaux, les dépendances, et lequel des trois environnements est touché ; il teste une hypothèse à la fois, en la nommant ; et il vous dit franchement ce qui reste inconnu au lieu d'inventer une cause. Les clés d'accès qu'il reconnaît dans les journaux qu'il vous montre arrivent masquées.

Les moments où on vous demande de choisir

Le harnais s'arrête à certains points et attend. Ce n'est pas de l'hésitation : ce sont les décisions qui vous appartiennent.

L'argent - toute dépense réelle, toute bascule d'un paiement en mode réel.

La destruction - supprimer une base, un projet, un dépôt.

La production - publier sur le site public. Pour deux gestes sans retour, écrire dans la base de production et changer le marché du projet, la confirmation se donne dans votre propre terminal, pas dans la conversation. Pour les autres, comme supprimer un projet ou passer

les paiements en réel, c'est votre réponse dans la conversation qui décide : lisez ce qu'on vous demande avant de répondre.

Le modèle - certaines étapes recommandent un modèle plus capable, donc plus cher, parce que l'enjeu le justifie. Au niveau d'autonomie par défaut, le harnais s'arrête et attend votre réponse : « go » si vous avez basculé, « continue » si vous gardez le modèle courant. À un niveau plus autonome, il vous le signale et poursuit.

Les arrêts les plus sensibles ne dépendent pas de la mémoire de la machine : ils sont inscrits dans une liste écrite des gestes sensibles, que les commandes consultent avant d'agir ; un geste absent de cette liste est traité comme une ligne rouge, jamais comme un oubli à interpréter.

Répondez à ces questions plutôt que de les contourner. Elles existent parce que quelqu'un a payé leur absence.

Quand la conversation devient longue

Une longue séance perd en précision : les décisions du début s'estompent. Le harnais le signale et propose une **passation** - un document qui résume l'état, les décisions prises et leurs raisons, ce qui reste à faire, plus un texte prêt à coller pour reprendre ailleurs. C'est `/handover` qui l'écrit. Les décisions durables n'y sont pas recopiées : elles vont au registre des décisions du projet, que la passation cite. Et une passation écrite le soir s'ajoute à celle du matin, au lieu de l'effacer.

Acceptez quand c'est proposé. Une passation s'écrit pendant que le contexte est encore net, pas quand il ne l'est plus.

Après une mise à jour du harnais

Il arrive qu'une séance ouverte avant la mise à jour vous serve encore l'ancien texte d'une commande. Sous Claude Code, la commande le dit avant d'agir, et indique le remède : ouvrir une nouvelle séance. Certaines mises à jour changent aussi les règles que le harnais installe dans votre assistant ; elles ne prennent effet qu'après avoir relancé `/start`, et le journal des versions le signale chaque fois.

Ce qui est clair sur le moment, et ce qui reste

Une bonne explication procure une sensation agréable : tout coule, tout paraît évident. Cette sensation ne dit rien de ce qui restera demain. On peut relire une page limpide et se découvrir incapable de la redire avec ses mots.

La règle tient en deux moitiés qui se contredisent en apparence.

Pour ce que vous devez SAVOIR - ce qu'est une base de test, pourquoi on ne publie pas directement - la difficulté est l'ennemie. Exigez la version simple. Une réponse qui vous laisse dans le flou est une réponse à refaire, et le dire n'a jamais coûté de temps à personne : `/explain` existe exactement pour ça.

Pour ce que vous devez SAVOIR FAIRE - lire un résultat, arbitrer un prix, valider une mise en ligne - la difficulté est l'outil. Faites-le vous-même une fois, maladroitement, plutôt que de regarder la machine le faire proprement plusieurs fois. L'effort de le produire laisse une trace ; regarder n'en laisse aucune.

Un test qui ne coûte rien : fermez l'écran et redites à voix haute ce que vous venez de lire. Si ça ne vient pas, vous avez lu, vous n'avez pas appris - et il vaut mieux le découvrir maintenant que le jour où il faut décider.

Le harnais s'applique la règle à lui-même. Les séances de `/prof` commencent par vous demander pourquoi vous apprenez, et ne notent pas un sujet comme acquis parce qu'il a été abordé : elles attendent que vous ayez fait ou dit quelque chose qui le prouve. Un relevé qui se remplirait tout seul ferait sauter, à la séance suivante, précisément ce que personne n'avait compris.

Ce que la machine ne fera pas à votre place

Elle ne sait pas si votre prix est le bon. Elle ne sait pas si vos clients préfèrent le mobile money à la carte. Elle ne sait pas si le nom que vous avez choisi sonne mal dans la langue de votre marché.

Elle construit vite et bien ce que vous décidez. **La décision reste la partie humaine**, et c'est celle qui fait la différence entre un produit qui marche et un produit qui existe.

ANNEXE

Le glossaire du projet

Cette annexe décrit un fichier que votre projet finira par porter, et qui ne ressemble à aucun autre : il ne contient ni code, ni plan, ni chiffres. Il contient **vos mots**.

Le problème qu'il règle

Vous dites « adhérent ». Votre associé dit « membre ». Le devis dit « client ». Ce sont les mêmes gens.

Tant que tout tient dans une conversation, personne ne s'en aperçoit. Le jour où un devis, un contrat et une page de vente circulent ensemble, le lecteur se demande s'il lui manque quelque chose - et un client qui doute d'un mot doute du reste.

Le harnais a longtemps eu ce défaut. Chaque séance d'entretien recueillait votre vocabulaire, s'en servait pour poser ses questions, puis **le jetait**. La séance suivante redemandait.

Ce que c'est, concrètement

Un fichier texte, dans le dossier des documents de votre projet, nommé `glossary.md`. Le nom du fichier est en anglais comme celui de tous les autres livrables ; ce qu'il y a dedans est dans votre langue et dans vos mots.

Chaque terme y tient en trois parties :

- Le mot, en titre - une ligne qui commence par deux dièses : `## Tontine`
- La définition, en une ou deux phrases. Les vôtres, pas celles d'un dictionnaire.
- Les mots écartés, sur une dernière ligne : `À éviter : cagnotte, groupe d'épargne`

La moitié qui a de la valeur

C'est la dernière ligne, celle des mots écartés.

Un glossaire qui ne porterait que des définitions vous apprendrait ce que vous savez déjà. Ce qui coûte cher n'est pas d'ignorer le sens d'un mot : c'est d'en avoir plusieurs pour la même chose. En retenir un et lister les autres comme à éviter, c'est ce qui rend un devis, un contrat et une page de vente cohérents entre eux - y compris quand ils sont écrits à six mois d'intervalle par deux personnes différentes.

Les mots écartés ne sont pas fautifs. Ils sont seulement les perdants d'un arbitrage que quelqu'un a fini par trancher, et qu'on ne veut plus rejouer.

Quand il se remplit

Jamais en avance. Un glossaire vide créé le premier jour est exactement le fichier que personne n'ouvre.

Il naît au premier mot qui le mérite, pendant une séance d'idéation (`/ideate`) ou de cahier des charges (`/spec`). Il s'enrichit ensuite quand vous butez sur un mot et demandez `/explain` : si ce mot appartient à votre métier et n'est pas encore écrit, on vous propose de l'inscrire.

Et s'il n'existe pas, aucune commande ne vous le reprochera. Son absence est un état sain, pas un retard.

Ce qu'on n'y met pas

Nos mots à nous. « Base de données », « déploiement », « serveur » sont des termes de notre métier, pas du vôtre. Vous avez le droit d'exiger qu'on vous les explique - c'est le travail de `/explain` - mais ils n'ont rien à faire dans ce fichier.

La barre est volontairement haute : un mot y entre quand **votre** métier lui donne un sens que le dictionnaire ne donne pas. Trois à cinq termes font déjà un glossaire complet. C'est en restant court qu'il garde son autorité, et c'est en s'allongeant qu'il devient un document que plus personne ne relit - donc un document qui finit par être faux sans que personne s'en aperçoive.

Qui le lit

Vous d'abord. C'est un fichier texte, et vous pouvez le corriger à la main le jour où une définition ne vous convient plus.

Le harnais ensuite : le cahier des charges et les explications le relisent avant d'écrire, pour employer vos mots et écarter les autres. C'est tout l'intérêt de l'avoir écrit une fois.

Pour finir

Vous avez la carte. Le reste est du travail, et le travail commence par une commande.

Si vous n'avez pas encore d'idée : `/pain-point-finder`. Si vous en avez une : `/ideate`, et acceptez qu'elle soit contestée. Si vous êtes déjà en route et perdu : `/next`, toujours.

La différence entre les gens qui livrent et les autres n'est pas le talent technique. C'est d'avoir commencé, puis de ne pas s'être arrêté à la première étape ennuyeuse - qui est souvent l'audit de sécurité, ou le plan d'acquisition, c'est-à-dire précisément celles qui décident si le produit vivra.

POUR ALLER PLUS LOIN

Par où commencer

Ce livre décrit le chemin. Le parcourir seul est possible : le harnais est fait pour ça, et quand vous ne saurez plus quoi faire, `/next` vous dira l'étape suivante et pourquoi c'est celle-là.

Des formations, en ligne et à Lomé

Pour qui préfère ne pas commencer seul. Les dates et les conditions sont sur dickoindustries.com/soka.

Une question, simplement

WhatsApp +228 96 64 92 92. C'est l'auteur de ces pages qui répond.

Vous n'avez besoin de rien de tout cela pour commencer. Vous avez la carte.

COLOPHON

Ce livre décrit Dicko Soka 3.169.64. Deuxième édition, fabriquée le 24 septembre 2026. La première édition, parue en août 2026, décrivait la version 3.86.0.

Tous les faits chiffrés de cet ouvrage - le nombre de commandes, la liste et l'ordre des 16 étapes du parcours, les noms des commandes citées - ont été **lus dans le harnais au moment de la fabrication**, et non recopiés à la main. Un contrôle automatique refuse de produire ce livre si une commande qu'il nomme n'existe pas, si une étape du parcours n'est pas traitée, ou si un chiffre y est écrit en dur.

Cela ne rend pas ce livre éternel : cela garantit qu'il était **exact pour la version nommée ci-dessus**. Le harnais a connu trois versions en une seule journée. Si la vôtre est plus récente, la commande `/about` vous dira laquelle vous avez, et une édition à jour de ce livre peut être refabriquée à partir des mêmes sources.

INVENTAIRE À CETTE VERSION

69	43	112
commandes publiques	rouages internes	au total

Édité par Dicko Industries.