ÉDITION NUMÉRIQUE · FRANÇAIS

Le nouveau *créateur* technique.

EXTRAIT GRATUIT

Introduction · Sommaire complet · Chapitre 2

AUTEUR

Hernán Capucci

Édition indépendante · 2026

Cet extrait inclut l'introduction, le sommaire complet et un chapitre représentatif du livre. L'objectif est de te permettre d'évaluer le ton, l'approche et le type d'apprentissage avant d'accéder à l'édition complète.

Ce que contient cet extrait

- L'introduction complète (Chapitre 0).
- Le sommaire complet du livre.
- Le Chapitre 2 en entier : *Comment fonctionne internet*.
- Des informations de continuité.

Sommaire complet

Chapitre 0 — Pourquoi ce livre existe

Un jour, tu demandes à l'intelligence artificielle de te générer quelque chose — du code pour une fonctionnalité, une automatisation qui relie deux services, un fragment qui résout un

problème précis. Ce qu'elle te renvoie semble fonctionner. Tu l'utilises. Et à un moment de ce processus surgit une question que tu n'oses peut-être pas poser à voix haute : comment je sais si c'est bien ?

Cette question n'est pas mineure. C'est la différence entre opérer avec ton propre discernement et opérer en espérant que personne ne trouve l'erreur.

Ce livre existe pour que tu puisses y répondre.

Accès sans compréhension

Il y a quelques années, construire un **logiciel** demandait une formation technique précise. Apprendre un langage de programmation, comprendre des structures de données, configurer des environnements de développement, lire une documentation qui supposait que tu savais déjà beaucoup. C'était une barrière haute et réelle.

Puis sont arrivés les outils d'**intelligence artificielle générative**. D'un coup, n'importe qui pouvait écrire une instruction en langage naturel et recevoir du code fonctionnel, une automatisation configurée, un système connecté à des services externes. La barrière technique a baissé de façon significative.

Mais c'est la barrière technique qui a baissé, pas la barrière conceptuelle.

Qu'un outil génère du code à ta place ne veut pas dire que tu comprends ce qu'il a généré. Qu'une plateforme déploie ton application en un clic ne veut pas dire que tu comprends ce qu'est un déploiement, ce qui peut lâcher ou comment revenir en arrière quand quelque chose tourne mal. Que l'IA t'explique une erreur ne veut pas dire que tu sais pourquoi elle s'est produite ni comment éviter qu'elle se reproduise.

L'accès s'est amélioré. La compréhension du **système** qui se trouve en dessous, dans bien des cas, n'a pas suivi le même rythme.

Créer n'est pas comprendre

Aujourd'hui, il y a plus de personnes qui construisent des produits numériques sans formation technique qu'à n'importe quel moment auparavant. Des applications, des automatisations, des intégrations entre services, des plateformes de contenu, des processus internes numérisés.

Cette démocratisation est réelle et elle a de la valeur. Le problème apparaît quand quelque chose ne marche pas comme prévu, ou quand il faut prendre des décisions sur le système qu'on a construit.

Un entrepreneur qui a utilisé l'IA pour construire sa boutique en ligne ne sait pas pourquoi « la base de données est tombée ». Il ne sait pas si l'information de ses clients est protégée. Il ne sait pas si le travail que lui a livré le développeur qu'il a engagé est bien fait ou non. Il ne sait pas quelles questions lui poser pour le savoir.

Un professionnel qui a automatisé ses processus avec l'IA ne comprend pas précisément ce que fait chaque partie de ce qu'il a monté. Il accepte le résultat parce que ça a marché hier. Quand ça cesse de marcher, il n'a pas les outils pour comprendre par où chercher.

Une fondatrice qui travaille avec une équipe technique externe acquiesce quand on lui parle d'**API**, de **dépôts** ou d'architecture du système. Mais elle ne comprend pas. Et elle ne demande pas, parce qu'elle ne sait pas quoi demander.

Dans tous ces cas, le problème n'est pas un manque d'intelligence ni de capacité. C'est un manque de vocabulaire et de cadre conceptuel — ce qui permet d'opérer dans l'écosystème numérique avec son propre discernement.

L'IA comme point de départ

C'est l'intelligence artificielle qui a rendu ce livre urgent. C'est le réveil-matin.

Mais la destination du livre n'est pas de parler d'intelligence artificielle. C'est de parler des systèmes qui existent sous n'importe quel projet numérique, avec ou sans IA.

Une API n'est pas un concept d'IA. Elle existe depuis des décennies. Un **dépôt** de code n'est pas nouveau non plus. La différence entre un environnement de développement et un environnement de production, ce n'est aucun modèle de langage qui l'a inventée. L'authentification des utilisateurs, les bases de données relationnelles, le cycle de vie d'un **déploiement** — tout cela existait avant l'IA générative et continuera d'exister après.

Ce qui a changé avec l'IA, c'est que ces choses sont maintenant à la portée de personnes qui, avant, n'y touchaient pas. Et cette accessibilité sans compréhension crée de nouvelles dépendances.

Dépendance à l'outil : s'il change, s'il lâche, s'il donne des résultats inattendus, tu n'as aucun moyen de les évaluer. Dépendance au développeur qui, lui, comprend : s'il change, s'il augmente ses tarifs, s'il disparaît, tu ne peux pas continuer sans lui. Dépendance à celui qui dit en savoir plus : sans cadre à toi, tu ne peux pas distinguer un vrai savoir de promesses creuses.

L'IA a ouvert une porte. Ce livre donne le vocabulaire pour comprendre ce qu'il y a derrière.

Le vocabulaire qui manque

Il y a une confusion fréquente qu'il vaut la peine de dissiper dès le départ.

Le problème que ce livre traite, ce n'est pas que tu ne saches pas programmer. Ne pas savoir programmer n'est pas le problème. Une énorme quantité de personnes très efficaces dans le monde numérique ne savent pas programmer et n'ont pas besoin de l'apprendre.

Ce qui compte, ce n'est pas savoir écrire du code. Ce qui compte, c'est savoir le lire avec assez de discernement pour comprendre ce qu'il fait. Ce qui compte, c'est comprendre ce qu'est une **base de données** pour pouvoir prendre des décisions sur la façon de stocker l'information. Savoir ce qu'est un dépôt pour pouvoir vérifier ce qu'un développeur t'a livré. Savoir ce qu'est un **endpoint** pour pouvoir demander à l'IA d'en construire un avec précision. Savoir ce qu'est un environnement de production pour ne pas casser quelque chose en direct par erreur.

Rien de tout cela n'exige d'écrire une seule ligne de code.

Imagine cette situation concrète : le développeur te dit « la migration a échoué dans l'environnement de staging ». Sans contexte, cette phrase est du bruit. Avec le vocabulaire de ce livre, tu comprends que « migration » est un changement dans la structure de la base de données, que « staging » est l'environnement de test avant la production, et que le problème n'a probablement pas encore touché les utilisateurs — mais qu'il faut le résoudre avant de continuer. Cette différence n'exige pas de programmer. Elle exige de comprendre les concepts.

Ce qu'elle exige, en revanche, c'est du vocabulaire technique. Le vocabulaire qui permet de poser les bonnes questions, d'interpréter les réponses, d'évaluer des propositions, de repérer des signaux d'alerte et de prendre des décisions avec discernement. Ce vocabulaire ne s'acquiert pas en utilisant des outils. Il s'acquiert en comprenant les concepts qui se trouvent derrière.

C'est ce que ce livre construit. Concept après concept, depuis zéro, avec des exemples universels et sans supposer aucune connaissance préalable.

À qui s'adresse ce livre

Ce livre est écrit pour les personnes qui construisent des choses numériques sans formation technique et qui veulent le faire avec plus de discernement.

Cela inclut celui qui monte son premier produit numérique avec l'aide de l'IA et ne comprend pas tout à fait ce qu'il est en train de construire. Celui qui travaille avec une équipe technique externe et veut pouvoir parler avec elle sans que chaque conversation soit un effort de traduction. Celui qui utilise des outils d'automatisation dans son travail quotidien et veut comprendre ce qu'il fait vraiment. Celui qui a une idée et a besoin de l'évaluer techniquement avant d'engager du temps et de l'argent pour la développer.

Peu importe le secteur ou le domaine de travail. Peu importe non plus l'âge ou le niveau d'éducation formelle en technologie.

Ce qui compte, en revanche, c'est que tu utilises l'ordinateur avec aisance dans ton travail ou ton projet. Que tu as essayé au moins un outil d'IA. Et que tu veux mieux comprendre l'**écosystème numérique** dans lequel tu opères.

Le point de départ peut être que tu n'as jamais ouvert un dépôt de code. Que tu n'as jamais configuré un environnement de développement. Que tu ne vois pas bien quelle est la différence entre un serveur et le cloud. Aucun de ces points de départ n'est un obstacle. Ce livre commence à partir de là.

Ce que tu ne trouveras pas

Avant d'aller plus loin, mieux vaut être explicite sur ce que ce livre ne fait pas.

Il n'apprend pas à programmer. Ce n'est un manuel d'aucun langage de programmation ni d'aucun outil spécifique. Si c'est ce que tu cherches, il existe des ressources meilleures et plus spécialisées pour ça.

Il n'affirme pas que l'IA fait tout. Ce n'est pas le cas. Elle a des capacités réelles et des limites réelles. Ce livre montre les deux avec honnêteté.

Il ne promet pas que dans un délai donné tu auras ton projet prêt. Les processus techniques ont leur propre complexité et leur propre temps. Les minimiser serait mentir.

Il ne dépend d'aucun outil, plateforme ou service spécifique. Quand un outil concret est mentionné, c'est à titre d'exemple. Les concepts enseignés s'appliquent de la même façon avec n'importe quel stack ou environnement que tu utilises.

Il ne parle pas de cas de réussite particuliers ni de projets réels comme modèles à suivre. Les exemples sont génériques et universels, par choix.

Il ne dit pas « c'est facile » quand ça ne l'est pas. Certains concepts de ce livre sont simples. D'autres demandent de l'attention et d'y revenir. Quand quelque chose a une complexité réelle, le livre le signale.

Ce que tu trouveras

En terminant ce livre, tu auras une compréhension concrète de concepts que tu ne connais peut-être aujourd'hui que de nom.

Tu comprendras ce qu'est une application de l'intérieur : quelle partie gère ce que tu vois à l'écran, quelle partie traite la logique et les données, et comment ces deux parties communiquent. Ça ne fera pas de toi quelqu'un capable de la construire seul, mais oui de comprendre de quoi on parle quand on te la décrit ou quand quelque chose tombe en panne.

Tu pourras ouvrir un dépôt de code et lire sa structure : quels dossiers existent, ce que fait chaque partie du projet, ce que dit l'historique des changements. Cette information est disponible dans n'importe quel dépôt et tu ne sais peut-être pas aujourd'hui que tu peux t'en servir pour comprendre ce que quelqu'un t'a livré.

Tu pourras écrire un **prompt** avec assez de contexte pour que l'IA produise quelque chose d'utile. La différence entre une instruction vague et une instruction précise n'est pas du talent : c'est savoir quelle information l'outil a besoin pour bien travailler.

Tu pourras vérifier ce que l'IA ou un développeur a généré avec des questions concrètes : que fait exactement cette partie ? Que se passe-t-il si ce champ arrive vide ? Y a-t-il une section qui accède à de l'information sensible ? Ces questions n'exigent pas de savoir programmer. Elles exigent de comprendre les concepts en jeu.

Tu pourras utiliser le glossaire de ce livre comme outil de travail : ouvrir une définition quand un terme inconnu apparaît, la comprendre, et revenir à la conversation ou au document avec plus de clarté qu'avant.

Et tu pourras reconnaître quand une réponse — d'une personne ou d'un outil — n'est pas évaluable avec tes connaissances actuelles. Ça ne veut pas toujours dire connaître la bonne réponse. Ça veut dire savoir quand la question que tu dois poser est meilleure que celle que tu as posée.

Comment il est organisé

Le livre a six parties en plus de ce chapitre introductif, des annexes pratiques et un glossaire technique.

La **Partie I** couvre les fondements : ce qu'est le logiciel, comment fonctionne internet, ce qu'est le modèle client-serveur, ce qu'est une base de données et ce qu'est le code. Ce sont les concepts les plus basiques de l'écosystème numérique. Chaque partie qui suit les tient pour acquis.

La **Partie II** entre dans l'architecture d'une application : le **frontend**, le **backend**, les API, les bases de données en profondeur, l'authentification et les sessions. C'est la structure interne de n'importe quel produit numérique moderne.

La **Partie III** couvre les outils et les processus qui font fonctionner un projet dans le monde réel : le contrôle de versions avec **Git**, le terminal de commandes, les environnements de développement, le déploiement et la sécurité de base.

La **Partie IV** est consacrée à travailler avec l'IA de façon intelligente : ce qu'elle peut faire, où elle échoue de manière systématique, comment formuler un prompt efficace, comment vérifier ce qu'elle produit, et comment utiliser des modèles, des **agents** et des automatisations sans perdre le contrôle du processus.

La **Partie V** réunit tout dans le contexte de construire quelque chose de concret : comment se préparer avant de commencer, comment communiquer avec des équipes techniques, comment passer d'une idée à un produit qui fonctionne.

La **Partie VI**, ce sont des annexes pratiques : des checklists de référence rapide, des commandes de base du terminal et de Git, des modèles de prompts réutilisables et des ressources pour approfondir chaque domaine.

À la fin du livre, il y a un **glossaire technique** avec plus de 200 termes expliqués en langage clair, avec des exemples et sans jargon. Ce n'est pas une annexe décorative. C'est une partie intégrante du livre.

L'ordre des parties n'est pas arbitraire. Chacune s'appuie sur la précédente. Les parties sur les systèmes et les outils ont besoin des fondements de la Partie I. Les parties sur l'IA et la construction ont besoin de l'architecture et des outils. Si tu choisis de lire dans l'ordre, c'est pour cette raison.

Comment l'utiliser

Il y a deux façons de lire ce livre et les deux sont valables.

La première, c'est d'une traite, du début à la fin. C'est celle qu'on recommande à qui a peu ou pas de contact préalable avec l'écosystème numérique. Le parcours construit la compréhension de manière cumulative : chaque chapitre prépare le terrain pour le suivant.

La seconde, c'est comme livre de référence. Quelqu'un t'a mentionné une API et tu n'as pas compris ce que c'est : tu ouvres le Chapitre 8. Tu vas faire ton premier déploiement et tu ne sais pas ce que ça implique : tu ouvres le Chapitre 14. Tu as besoin de comprendre ce qu'est l'**authentification** avant une réunion sur la sécurité : tu ouvres le Chapitre 10. Chaque chapitre peut se lire de manière indépendante, même si les renvois indiquent ce qu'il vaut mieux avoir lu avant.

Le glossaire fonctionne de la même manière. Quand un terme technique apparaît pour la première fois dans un chapitre, il est en **gras** avec un renvoi au glossaire. Dans la version numérique du livre, ce renvoi est un hyperlien actif : tu peux aller directement à la définition et revenir au chapitre en un clic. Dans la version imprimée, le renvoi indique le numéro de page.

Une note sur les exemples : tout au long du livre, tu vas retrouver les mêmes types de contextes — une boutique en ligne, une application de gestion de réservations, une plateforme de cours. Ce sont des situations génériques choisies parce qu'elles n'exigent aucune connaissance préalable d'aucun secteur. Les concepts s'appliquent exactement de la même façon à n'importe quel autre type de projet.

Une note sur les outils : quand un outil est mentionné comme exemple, c'est ça, un exemple. Les concepts que ce livre enseigne ne dépendent d'aucune plateforme spécifique. Si l'outil que tu utilises aujourd'hui change demain, les concepts restent valables.

Créer avec son propre discernement

Il y a une différence entre quelqu'un qui utilise des outils numériques sans les comprendre et quelqu'un qui les utilise en sachant ce qu'il fait.

Cette différence n'est pas dans le fait de savoir programmer. Elle est dans le fait d'avoir le vocabulaire pour poser les bonnes questions. Le cadre conceptuel pour évaluer les réponses. La capacité de détecter quand quelque chose ne va pas même sans savoir exactement pourquoi. L'autonomie de prendre des décisions sans dépendre entièrement des autres pour comprendre ce qui se passe.

C'est le profil que ce livre construit : le nouveau créateur technique. Quelqu'un qui utilise l'IA, travaille avec des équipes techniques, construit des choses numériques — et le fait avec son propre discernement.

Ce n'est pas un programmeur. Il ne prétend pas l'être. Mais il n'opère pas à l'aveugle non plus, n'accepte pas des résultats sans pouvoir les évaluer et ne dépend pas qu'un autre lui explique tout.

Il comprend le système dans lequel il travaille. Il sait quoi demander, comment évaluer ce qu'il reçoit et quand reconnaître que quelque chose ne va pas. Cette compréhension n'exige pas d'écrire du code. Elle exige de comprendre comment fonctionne ce que tu utilises.

Ce que tu as appris dans ce chapitre

- L'IA a démocratisé l'accès à l'exécution technique, mais pas la compréhension de l'écosystème numérique.
- Ne pas savoir programmer n'est pas le problème. Le manque d'un vocabulaire technique minimal, lui, l'est.
- Ce livre n'apprend pas à programmer. Il enseigne les concepts qui permettent de créer avec discernement, de parler avec des équipes techniques et de vérifier ce que produit l'IA.
- Le livre a six parties, des annexes pratiques et un glossaire de plus de 200 termes. Il peut se lire d'une traite ou s'utiliser comme référence.
- Dans la version numérique, les termes du glossaire sont des hyperliens actifs dans chaque chapitre.

Ce qui suit

Le Chapitre 1 commence par le début : ce qu'est le logiciel et pourquoi il importe de le comprendre même si tu ne vas jamais l'écrire. C'est le premier bloc de la Partie I et la base de tout ce qui vient après.

Chapitre 2 — Comment fonctionne internet

Tu écris une adresse dans le navigateur et tu appuies sur Entrée. En moins d'une seconde, une page apparaît : du texte, des images, des données à jour.

D'où est venu tout ça ? Comment est-ce arrivé jusqu'à ton écran ? Et pourquoi, parfois, ça n'arrive pas ?

Ces questions n'exigent pas de savoir programmer pour y répondre. Elles exigent de comprendre l'infrastructure qui relie n'importe quel appareil à n'importe quel serveur dans le monde. Cette compréhension est la base pour pouvoir diagnostiquer pourquoi quelque chose marche — et pourquoi, parfois, ça ne marche pas.

Le trajet d'une requête

Quand tu écris une adresse dans le navigateur, ce que tu écris s'appelle une **URL** — Uniform Resource Locator, ou localisateur uniforme de ressource. C'est l'adresse qui identifie une ressource précise sur internet : une page, une image, un fichier, une donnée.

Le navigateur a besoin de trouver le serveur qui possède cette ressource. Mais les serveurs ne s'identifient pas par leurs noms lisibles : ils s'identifient par des numéros. Chaque serveur a une **adresse IP** — une suite de chiffres qui le situe de manière unique sur le réseau, un peu comme une adresse postale situe un bâtiment dans une ville.

Le problème, c'est que toi, tu n'écris pas des numéros. Tu écris `www.encyclopedie.org` ou `actus.exemple.com`. Pour que le navigateur puisse trouver le serveur correspondant, il doit traduire ce nom en une adresse IP. Ce processus s'appelle la résolution **DNS** — Domain Name System, ou système de noms de domaine.

Le DNS fonctionne comme l'annuaire téléphonique d'internet. Le navigateur interroge un serveur DNS et lui demande : « quelle IP a `actus.exemple.com` ? » Le serveur DNS répond avec le numéro. Le navigateur a maintenant l'adresse réelle.

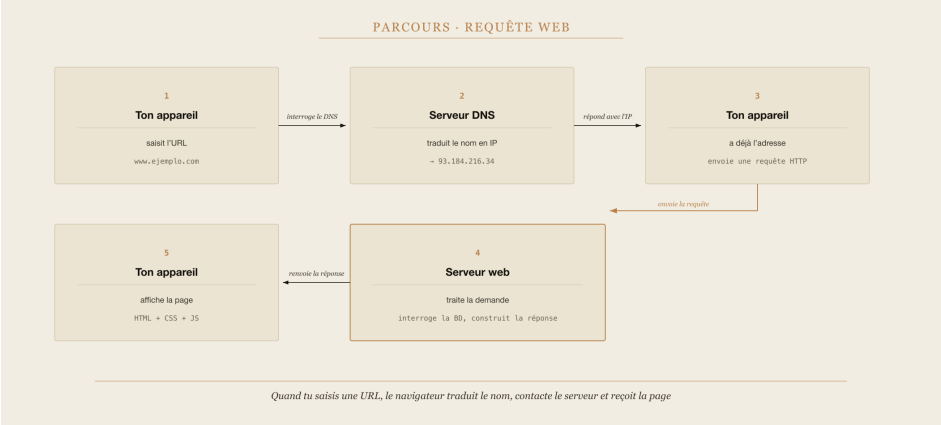

Fig. 1. Quand tu écris une adresse, le navigateur ne va pas vers un cloud magique : il déclenche une chaîne de traductions, de demandes et de réponses.

Avec cette adresse, le navigateur envoie une requête au serveur. Cette requête voyage à travers le réseau — par des câbles, des fibres optiques et des connexions sans fil — jusqu'à atteindre le serveur qui l'attend. Le serveur la reçoit, traite ce qu'il a à traiter, et renvoie une réponse. Cette réponse voyage en sens inverse jusqu'au navigateur. Le navigateur interprète le contenu et l'affiche à l'écran.

Tout ce processus — requête DNS, voyage aller, traitement, réponse retour — se produit en fractions de seconde dans des conditions normales.

Il y a une optimisation qui vaut la peine d'être mentionnée : le **cache**. Le navigateur n'interroge pas le DNS à chaque fois que tu visites le même site. Il garde la réponse — la traduction du nom en IP — pendant un certain temps. Le système d'exploitation fait de même. Ça rend les visites répétées plus rapides. Mais ça signifie aussi que si le serveur a récemment changé d'adresse IP, ton appareil utilise peut-être encore l'ancienne adresse. Vider le cache du navigateur ou attendre qu'il expire résout ce problème.

Quand tu écris une adresse, le navigateur ne va pas vers un cloud magique : il déclenche une chaîne de traductions, de demandes et de réponses.

Ce qu'est un domaine

Un **domaine** est le nom lisible qui identifie un site ou un service sur internet. `encyclopedie.org`, `actus.exemple.com`, `monapp.io` sont des domaines.

Les domaines n'existent pas tout seuls : quelqu'un les enregistre. Il y a des entreprises spécialisées dans l'enregistrement de domaines — on les appelle des bureaux d'enregistrement — et

le domaine se loue par périodes, en général un an. S'il n'est pas renouvelé, le domaine expire et redevient disponible pour qu'une autre personne l'enregistre.

La partie finale du domaine — `.com`, `.org`, `.io`, `.fr` — s'appelle le domaine de premier niveau ou TLD (Top Level Domain). Elle indique le type d'organisation ou le pays d'origine. Cette distinction a une valeur historique et de positionnement, mais techniquement elle ne change pas le fonctionnement du système.

La partie qui peut prêter à confusion, c'est la relation entre domaine et serveur. Un domaine n'est pas le serveur : c'est l'étiquette qui pointe vers le serveur. Un même serveur peut avoir plusieurs domaines qui pointent vers lui. Un même domaine peut rediriger vers différents serveurs selon le moment de la journée ou la région du monde depuis laquelle on y accède. Le DNS est le système qui maintient à jour cette relation entre noms et adresses.

Quand un domaine n'est pas configuré correctement — ou quand il a expiré — le DNS ne peut pas résoudre l'adresse. Le navigateur ne sait pas vers quel serveur aller. La page ne charge pas, même si le serveur qui la contient fonctionne parfaitement.

Ce qu'est un serveur — et ce qu'est l'hébergement

Un **serveur** est un ordinateur conçu pour recevoir des requêtes et renvoyer des réponses. Sur le fond, il n'est pas différent de n'importe quel ordinateur : il a un processeur, de la mémoire, du stockage et une connexion au réseau.

La différence pratique, c'est qu'un serveur est allumé en permanence, connecté à internet avec une haute disponibilité, et configuré pour traiter plusieurs requêtes simultanées. Il n'a ni écran ni clavier. Il n'est sur le bureau de personne. Il vit dans un centre de données — une installation conçue pour maintenir de nombreux serveurs en fonctionnement de manière continue et sécurisée.

L'**hébergement** est le service qui te loue de l'espace et des ressources sur ces serveurs. Quand tu « publies » quelque chose sur internet — une page, une application, un fichier — ce que tu fais, c'est placer ce contenu sur un serveur d'une entreprise d'hébergement, configurer un domaine pour qu'il pointe là, et laisser le serveur le livrer à qui le demande.

Il existe différentes formes d'hébergement. Dans la plus basique, tu partages le serveur avec d'autres sites — c'est moins cher mais avec des ressources limitées. Dans des formes plus avancées, tu as un serveur virtuel ou physique exclusif — plus cher mais avec plus de contrôle et de capacité. Dans le modèle cloud, les ressources s'attribuent dynamiquement selon la demande.

Pour qui construit quelque chose de numérique, comprendre ce qu'est l'hébergement signifie comprendre que le système que tu as créé vit quelque part physiquement, que cet endroit a des limites de capacité, et que si cet endroit lâche — ou si le domaine ne pointe pas correctement vers lui — le système n'est pas accessible, même s'il est parfaitement construit.

Une distinction utile : le serveur web est le composant qui reçoit les requêtes HTTP et renvoie du contenu. Le serveur d'application est celui qui exécute la logique du système — ce qu'on verra plus en détail au Chapitre 3. Dans beaucoup de systèmes simples, les deux fonctions tournent sur la même machine. Dans des systèmes plus grands, elles sont séparées. Ce qui importe pour l'instant, c'est que « le serveur » n'est pas une seule chose monolithique : c'est un ensemble de pièces qui travaillent ensemble pour répondre aux requêtes.

HTTP et HTTPS : la langue du web

Quand le navigateur envoie une requête et que le serveur renvoie une réponse, tous les deux ont besoin de parler la même langue. Cette langue s'appelle un **protocole**. Un protocole est un ensemble de règles qui définit comment les messages sont formatés, transmis et interprétés.

Le protocole du web est **HTTP** — HyperText Transfer Protocol. Il définit la structure des requêtes et des réponses : quelle information vient en premier, comment on indique le type de contenu, comment on communique le résultat de l'opération.

Une requête HTTP de base inclut : la méthode (ce que tu veux faire — obtenir de l'information, envoyer des données, supprimer quelque chose), l'adresse de la ressource, et des en-têtes qui contiennent de l'information supplémentaire sur qui demande et quel type de réponse il accepte. Une réponse HTTP inclut : le code de statut (si ça a marché ou non), des en-têtes avec de l'information sur le contenu, et le corps — le contenu lui-même.

HTTPS est la version sécurisée de HTTP. Le « S » est pour « Secure ». La différence technique, c'est qu'en HTTPS toute la communication voyage chiffrée : les données que le navigateur envoie au serveur et celles que le serveur renvoie ne peuvent pas être lues par quelqu'un qui intercepterait la connexion en chemin.

Ce chiffrement est rendu possible par un **certificat SSL/TLS** — un fichier numérique que le serveur présente au navigateur pour prouver qu'il est bien qui il prétend être, et qui active le chiffrement de la communication. Les navigateurs modernes affichent un cadenas dans la barre d'adresse quand la connexion utilise HTTPS.

Pourquoi ça importe pour qui construit : tout système qui manipule des données d'utilisateurs — mots de passe, information personnelle, paiements — doit utiliser HTTPS sans exception. Sans HTTPS, ces données voyagent en clair sur le réseau et peuvent être capturées. Avec HTTP, le navigateur peut afficher un avertissement « site non sécurisé » qui éloigne les utilisateurs et réduit la confiance dans le système.

> ***Concept clé : protocole*** *Ensemble de règles qui définit comment deux parties communiquent. HTTP est le protocole qu'utilisent les navigateurs et les serveurs pour échanger de l'information sur le web. HTTPS est sa version chiffrée.*

Ce que le serveur répond

Chaque fois qu'un serveur reçoit une requête, il renvoie une réponse qui inclut un numéro : le **code de statut**. Ce numéro indique ce qui est arrivé à la requête.

Les codes se regroupent par catégories. Ceux qui commencent par 2 signifient le succès. Ceux qui commencent par 3 sont des redirections. Ceux qui commencent par 4 indiquent une erreur du côté du client — quelque chose dans la requête ne va pas. Ceux qui commencent par 5 indiquent une erreur du côté du serveur — le serveur a reçu la requête mais n'a pas pu la traiter correctement.

Les trois plus importants pour qui travaille avec des systèmes :

200 — OK. La requête a été traitée correctement et le serveur a renvoyé le contenu attendu. C'est le résultat normal. Si tu vois une page, le serveur a répondu 200.

404 — Not Found. Le serveur a reçu la requête, mais la ressource que tu as demandée n'existe pas sur ce serveur. L'adresse que tu as utilisée ne correspond à aucun fichier, page ou donnée disponible. Ce n'est pas une erreur du serveur : c'est que ce que tu cherchais n'est pas là.

500 — Internal Server Error. Le serveur a reçu la requête, a tenté de la traiter, et quelque chose a lâché dans son propre fonctionnement. Le problème n'est pas dans ce que tu as demandé : il est dans la façon dont le serveur a géré la demande. C'est une erreur du code ou de la configuration du serveur.

Code	Nom	Ce qui s'est passé	Exemple courant
200	OK	Le serveur a trouvé et livré ce que tu as demandé.	Tu ouvres une page et tout charge sans problème.
301 / 302	Redirection	La ressource est à une autre adresse. Le navigateur t'y emmène automatiquement.	Une ancienne URL te renvoie vers le nouveau domaine du site.
400	Bad Request	La requête est arrivée mal formée ; le serveur n'a pas pu l'interpréter.	Tu envoies un formulaire avec des données obligatoires incomplètes.
401	Unauthorized	Tu n'as pas présenté d'identifiants valides. Le serveur ne sait pas qui tu es.	Tu tentes de voir du contenu privé sans te connecter.
403	Forbidden	Tes identifiants sont valides, mais tu n'as pas la permission pour ça.	Tu es entré dans ton compte, mais cette section ne correspond pas à ton rôle.
404	Not Found	Ce que tu as demandé n'existe pas sur ce serveur.	Une URL a été mal écrite ou n'existe plus.
500	Internal Server Error	Le serveur a reçu ta demande mais a échoué à la traiter.	Tu tentes de finaliser un achat et « erreur inattendue » apparaît.

Code	Nom	Ce qui s'est passé	Exemple courant
503	Service Unavailable	Le serveur ne peut pas répondre maintenant ; il est saturé ou en maintenance.	Un site tombe pendant une vente massive ou un lancement.

Pas besoin de mémoriser tous les codes. L'important, c'est de reconnaître le motif : les 2xx indiquent en général le succès, les 3xx une redirection, les 4xx un problème du côté de la requête et les 5xx un problème du côté du serveur.

Pourquoi « l'app est tombée »

« La page ne charge pas », « le système est tombé », « ça ne marche pas » — quand quelque chose cesse d'être disponible, il existe en général une cause technique précise. Comprendre les points de défaillance possibles permet de circonscrire le problème avant d'appeler quelqu'un pour le résoudre.

Les causes les plus courantes, classées par l'endroit où elles surviennent :

Le serveur est tombé. Le serveur a cessé de répondre. Ça peut être parce que le service d'hébergement a eu un problème, parce que le serveur a redémarré pour une mise à jour, ou parce que le système s'est retrouvé sans ressources (mémoire, CPU) et a cessé de fonctionner. Dans ce cas, le navigateur ne reçoit aucune réponse d'aucune sorte.

Le domaine ne résout pas. Le DNS ne peut pas traduire le nom du domaine en une adresse IP. Ça peut être parce que le domaine a expiré, parce que la configuration du DNS a mal changé, ou parce qu'il y a un problème avec le serveur DNS. Le résultat ressemble au précédent : le navigateur ne sait pas où aller.

Le certificat a expiré. Si le certificat HTTPS du serveur a expiré, le navigateur bloque la connexion et affiche un avertissement de sécurité. La page existe techniquement et le serveur fonctionne, mais le navigateur protège l'utilisateur en refusant la connexion. Le résultat visible : un écran d'avertissement, pas la page.

Erreur 500. Le serveur répond, mais avec une erreur interne. La page existe, le serveur est allumé, mais le code a rencontré un problème en traitant la requête. L'utilisateur voit un message d'erreur au lieu du contenu attendu.

Problème de réseau. La connexion entre l'appareil de l'utilisateur et le serveur est interrompue à un point intermédiaire. Ça peut être la connexion internet de l'utilisateur, un fournisseur réseau, ou un point de l'infrastructure entre les deux côtés.

Les distinguer superficiellement aide à savoir par où commencer :

- Si la page ne charge rien et qu'il n'y a pas d'avertissement → possiblement serveur tombé ou DNS qui ne résout pas.
- Si le navigateur affiche un avis de sécurité explicite → certificat expiré ou HTTP sans chiffrement.
- Si la page charge avec un message d'erreur dans le contenu → probablement erreur 500, problème du code.
- Si seul toi ne peux pas accéder mais les autres oui → possiblement un problème de réseau ou de cache local.

Savoir cela ne veut pas dire pouvoir le résoudre directement. Ça veut dire pouvoir le communiquer avec précision à qui, lui, peut le faire.

> ***Questions utiles quand un site ne charge pas***
>
> *Quand le système n'est pas disponible, ces questions aident à circonscrire le problème avant d'appeler qui peut le résoudre :*
>
> - *Le problème arrive-t-il à tout le monde ou seulement à moi ?*
> - *Le domaine résout-il ? Le nom pointe-t-il vers une adresse ?*
> - *Le serveur répond-il ? Reçoit-on quelque chose, ne serait-ce qu'une erreur ?*
> - *Quel code d'erreur y a-t-il — 404, 500 — ou n'y a-t-il aucune réponse ?*
> - *Est-ce la production qui lâche ou seulement un environnement de test ?*
> - *Y a-t-il des logs de l'erreur ?*
> - *Y a-t-il eu un changement récent — déploiement, changement de configuration, renouvellement de domaine ?*

Il y a un cas qui vaut la peine d'être mentionné parce qu'il est fréquent et déroutant : le système fonctionne pour certaines personnes et pas pour d'autres. Cela ne signifie presque jamais que

le serveur est tombé. En général, ça indique un problème de propagation DNS — quand un changement a été fait dans la configuration de domaine, ce changement met entre quelques minutes et quelques heures à atteindre tous les serveurs DNS du monde. Pendant ce temps, différents utilisateurs interrogent différents serveurs DNS et obtiennent différentes réponses. Certains voient le nouveau site ; d'autres voient encore l'ancien ou ne voient rien. C'est un comportement attendu, pas une erreur à résoudre en urgence.

Comment tu vas l'entendre en réunion

Dans une réunion avec une équipe technique ou avec un prestataire, on utilise des phrases qui supposent que tout le monde comprend de quoi on parle. Voici les plus courantes sur ce sujet, ce qu'elles signifient en pratique, et ce que tu peux demander.

« **On dirait un problème de DNS.** » Le nom du domaine ne résout pas correctement — il ne peut pas être traduit en l'adresse IP du serveur. Ça peut être une configuration incorrecte ou un changement récent qui n'a pas encore complètement propagé.

Question utile : « Est-ce que ça arrive à tous les utilisateurs ou seulement à certains ? » Si c'est à tous, c'est probablement une erreur de configuration. Si c'est seulement à certains, le plus probable est que ce soit une propagation en cours — un processus qui met des heures et se résout tout seul.

Mieux vaut ne pas supposer : que le serveur est tombé. Le DNS et le serveur sont des pièces distinctes. Le serveur peut fonctionner parfaitement pendant que le DNS lâche.

« **Le serveur répond 500.** » Le serveur a reçu la requête mais a rencontré une erreur en la traitant. Le problème est dans le code ou dans la configuration du système, pas dans la requête de l'utilisateur ni dans le réseau.

Question utile : « Y a-t-il des logs de l'erreur ? » Les logs sont le registre de ce qu'a fait le système avant de lâcher. Un 500 laisse presque toujours une trace qui identifie la cause. Sans logs, le diagnostic devient bien plus difficile.

Mieux vaut ne pas supposer : que c'est un problème d'internet ou de l'appareil de l'utilisateur. Un 500 signifie que le serveur a effectivement répondu — seulement avec une erreur interne.

« **Le certificat a expiré.** » Le certificat HTTPS du serveur a expiré. Le navigateur détecte que la connexion ne peut plus être garantie comme sûre et bloque l'accès par défaut. Le système peut fonctionner parfaitement à l'intérieur, mais aucun utilisateur ne peut entrer.

Question utile : « Combien de temps prend le renouvellement ? » Dans la plupart des cas, c'est un processus rapide. Ce qui importe de savoir, c'est le temps estimé de résolution et s'il y a déjà des utilisateurs touchés qu'il faut prévenir.

Mieux vaut ne pas supposer : qu'il y a un problème de code ou d'infrastructure. C'est une expiration administrative, comme un document qui arrive à échéance. Ça n'indique rien sur la qualité du système.

« **C'est tombé en production.** » Le système qu'utilisent les vrais utilisateurs n'est pas disponible. Ce n'est pas un environnement de test ni de développement : c'est celui qui a le trafic réel et les données réelles. Cette phrase indique l'urgence.

Question utile : « Est-ce qu'on travaille déjà à le résoudre ? Quel est le temps estimé ? » Ces deux questions donnent le contexte nécessaire pour savoir s'il faut communiquer quelque chose aux utilisateurs ou attendre en silence.

Mieux vaut ne pas supposer : que le problème touche tout le monde de la même façon. Parfois la panne est partielle — elle ne touche qu'une région, un appareil ou une fonctionnalité précise.

Ce que tu as appris dans ce chapitre

- Une URL est l'adresse qui identifie une ressource sur internet. Pour l'atteindre, le navigateur interroge le DNS, qui traduit le nom du domaine en une adresse IP.
- Un serveur est un ordinateur toujours allumé qui reçoit des requêtes et renvoie des réponses. L'hébergement est le service qui loue ce serveur.
- HTTP est le protocole qui définit comment communiquent le navigateur et le serveur. HTTPS est sa version chiffrée, obligatoire pour tout système qui manipule des données sensibles.
- Les codes de statut sont le langage qu'utilise le serveur pour indiquer ce qui s'est passé : 200 c'est le succès, 404 c'est ressource non trouvée, 500 c'est erreur interne du serveur.
- « L'app est tombée » a des causes techniques précises : serveur tombé, DNS qui ne résout pas, certificat expiré, erreur de code. Les connaître permet de circonscrire le problème.

Ce qui suit

Le Chapitre 3 entre dans les deux acteurs de cette conversation que tu viens de comprendre : le client et le serveur. Tu sais déjà comment l'information voyage entre eux. Ce qui suit, c'est de comprendre ce que fait chacun dans cet échange — et pourquoi cette division est la structure fondamentale de n'importe quelle application.

Cet extrait s'arrête ici.

Le livre complet poursuit avec la carte complète :

client, serveur, bases de données, code,

API, sécurité, intelligence artificielle et déploiement.

L'édition complète comprend

22 chapitres, des annexes pratiques,

un glossaire technique et les versions PDF + EPUB.