GUIDE COMPLET · ÉDITION 2026

Les Métiers du Développement Logiciel

Comprendre chaque spécialisation d'une équipe technique — Frontend, Backend, Full Stack, DevOps et au-delà — comme dans une grande entreprise. Un parcours pédagogique pensé pour les grands débutants.

Frontend | Backend | Full Stack | DevOps | + 12 autres rôles

Manuel de formation

Du besoin métier à la mise en production

Niveau : Débutant

Aucun prérequis technique

Table des matières

MODULE 0

Introduction : comprendre le terrain de jeu

Avant de parler de métiers, posons les fondations. À la fin de ce module, vous saurez ce qu'est « développer », ce qui se passe quand vous ouvrez un site, et qui fait quoi dans une équipe.

Imaginez une grande cuisine de restaurant. Certains accueillent les clients et dressent les assiettes (ce qui se voit), d'autres préparent les plats en coulisses (ce qui ne se voit pas), un chef coordonne le tout, et une équipe s'assure que les fourneaux, le gaz et la livraison fonctionnent. Le développement logiciel, c'est exactement cela : une cuisine où chaque rôle a sa place. Ce guide vous présente chaque poste, un par un.

0.1 — Qu'est-ce que « développer » un logiciel ?

Développer, c'est écrire des instructions qu'un ordinateur peut exécuter pour résoudre un problème humain : afficher une page, enregistrer une commande, envoyer un message. On écrit ces instructions dans un **langage de programmation** (Python, JavaScript, etc.), un peu comme on écrirait une recette très précise que la machine suit à la lettre, sans rien deviner.

> **DÉFINITION — CODE SOURCE**
>
> Le **code source** est l'ensemble des fichiers texte écrits par les développeurs. C'est la « recette » du logiciel. On le transforme ensuite (compilation ou interprétation) en quelque chose que la machine exécute réellement.

Un « logiciel » peut prendre plusieurs formes : un **site web** (consulté dans un navigateur), une **application mobile** (sur téléphone), une **application de bureau**, ou encore un **service invisible** qui tourne sur des serveurs et rend service à d'autres programmes (une API). Dans ce guide, nous prenons surtout l'exemple des applications web, car c'est là que toutes les spécialisations se rencontrent le plus clairement.

0.2 — Le voyage d'une requête : ce qui se passe quand vous ouvrez un site

C'est l'exemple le plus utile pour comprendre qui fait quoi. Suivons ce qui arrive quand vous tapez l'adresse d'une boutique en ligne et appuyez sur Entrée.

1. Le navigateur cherche l'adresse

Votre navigateur demande au **DNS** (l'annuaire d'Internet) : « à quelle adresse IP correspond ce nom de site ? ». Il obtient un numéro, comme on cherche un numéro de téléphone.

2. La requête voyage jusqu'au serveur

Le navigateur envoie une requête **HTTP(S)** (« donne-moi la page d'accueil ») à travers Internet jusqu'au **serveur** de la boutique. C'est le domaine du **réseau** et du **DevOps**.

3. Le serveur réfléchit (Backend)

Le code **backend** reçoit la demande, vérifie qui vous êtes, va chercher les produits dans la **base de données**, applique les règles métier (promotions, stock) et prépare une réponse.

4. La réponse revient et s'affiche (Frontend)

Le navigateur reçoit du **HTML, CSS et JavaScript**. Le code **frontend** transforme tout cela en une belle page : images, boutons, animations. C'est ce que vous voyez et touchez.

5. Vous interagissez en boucle

Vous cliquez « Ajouter au panier » → une nouvelle petite requête part vers le backend → la réponse met à jour l'écran. Et ainsi de suite, des centaines de fois.

À RETENIR

Tout repose sur un dialogue **client ↔ serveur**. Le **client** (votre navigateur) demande ; le **serveur** répond. Le **Frontend** s'occupe du client (ce qui se voit), le **Backend** du serveur (ce qui se calcule), le **DevOps** de la route et des machines entre les deux, et le **Full Stack** touche aux deux côtés.

0.3 — L'organigramme d'une équipe technique en grande entreprise

Dans une grande entreprise, on ne mélange pas les rôles : chaque tâche précise est confiée à une spécialisation. Voici comment les équipes s'emboîtent, du plus proche du code jusqu'à la direction.

Niveau / Pôle	Qui s'y trouve et ce qu'il fait
Direction technique	**CTO** (vision et stratégie technique), **VP Engineering** (organisation des équipes), **Architecte logiciel** (choix techniques structurants).
Pilotage produit & projet	**Product Manager / Product Owner** (le « quoi » et le « pourquoi »), **Scrum Master** (fluidité de l'équipe), **UX/UI Designer** (l'expérience et le visuel).
Équipe de développement	**Tech Lead** (référent technique de l'équipe), développeurs **Frontend**, **Backend**, **Full Stack**, **Mobile**.
Données & intelligence	**Data Engineer** (tuyauterie de la donnée), **Data Scientist / Analyst** (analyse), **ML / AI Engineer** (modèles d'IA).
Plateforme & exploitation	**DevOps**, **Platform / Cloud Engineer**, **SRE** (fiabilité), **DBA** (bases de données), **Ingénieur Sécurité / DevSecOps**.
Qualité	**QA Engineer / Testeur** (manuel et automatisé), garant que rien ne casse avant la livraison.

ASTUCE DE LECTURE

Dans une **petite startup**, une seule personne porte souvent plusieurs casquettes (un Full Stack qui fait aussi un peu de DevOps). Dans une **grande entreprise**, chaque casquette devient un métier à part entière avec ses experts. Ce guide décrit la version « grande entreprise », où chaque tâche a sa spécialisation.

0.4 — Le vocabulaire de survie

Une dizaine de mots reviennent partout. Gardez cette page sous la main ; tout le reste du guide devient plus simple une fois ces mots apprivoisés.

Terme	En français simple
Serveur	Un ordinateur puissant, allumé en permanence, qui « sert » des données à d'autres machines.
Client	Le programme qui demande (votre navigateur, une appli mobile).
API	Un « menu » de commandes qu'un programme propose aux autres pour échanger des données. Le contrat entre frontend et backend.
Base de données	Le « grand classeur » organisé où l'on range durablement les informations (comptes, commandes…).
Framework	Une boîte à outils toute prête qui évite de tout réécrire (ex. React, Django).
Bibliothèque (library)	Un ensemble de fonctions réutilisables que l'on ajoute à son projet.
Bug	Un comportement non voulu du logiciel ; une erreur à corriger.
Déploiement	L'action de publier le logiciel pour que les vrais utilisateurs puissent l'utiliser.
Production (« la prod »)	L'environnement réel, en ligne, utilisé par les vrais clients. À ne pas casser !
Git / dépôt (repo)	L'outil qui sauvegarde l'historique du code et permet de travailler à plusieurs sans s'écraser.

Vous avez maintenant la carte du territoire. Entrons dans chaque métier, en commençant par celui que tout le monde voit en premier : le Frontend.

MODULE 1 — SPÉCIALISATION

Le Développeur Frontend — la vitrine

Tout ce que l'utilisateur voit, lit, clique et ressent. Le Frontend construit l'interface : la beauté, la clarté et la fluidité de l'expérience.

1.1 — En une phrase

Le **développeur Frontend** (ou « développeur front », ou « développeur client ») construit la partie **visible** d'une application : ce qui s'affiche dans le navigateur ou l'écran. Son obsession : que l'interface soit belle, rapide, compréhensible et utilisable par tout le monde, sur tout appareil.

> **DÉFINITION — FRONTEND**
>
> Le **frontend** (« devant ») désigne le code qui s'exécute du côté de l'utilisateur, dans le navigateur. C'est l'opposé du **backend** (« derrière »), qui tourne sur le serveur. Métaphore : au restaurant, le frontend est la salle et le service ; le backend est la cuisine.

1.2 — Ses missions précises (qui font son métier)

Dans une grande entreprise, voici les tâches concrètes confiées au Frontend, et à lui seul :

- **Intégrer les maquettes** : transformer le design fourni par l'UX/UI Designer (souvent dans Figma) en pages réelles, fidèles « au pixel près ».
- **Construire les composants d'interface** : boutons, formulaires, menus, fenêtres, listes, tableaux, réutilisables partout dans l'application.
- **Gérer l'interactivité** : ce qui se passe quand on clique, tape, fait défiler, glisse-dépose — sans recharger la page.
- **Connecter l'interface au backend** : appeler les **API** pour récupérer ou envoyer des données, puis afficher le résultat (et gérer les erreurs et les chargements).
- **Rendre le site « responsive »** : s'assurer qu'il s'adapte aussi bien au téléphone qu'à l'écran d'ordinateur.
- **Garantir l'accessibilité** : rendre l'interface utilisable par les personnes en situation de handicap (lecteurs d'écran, navigation au clavier, contrastes).
- **Optimiser la performance** : pages qui se chargent vite, images légères, animations fluides.
- **Tester et corriger** : écrire des tests d'interface et corriger les bugs d'affichage.

1.3 — La journée type

Matin

Réunion debout de 15 min (« daily ») : ce que je fais aujourd'hui, mes blocages. Puis intégration d'une nouvelle page validée par le designer.

Après-midi

Connexion de la page à l'API du backend, gestion des cas « chargement » et « erreur », relecture du code d'un collègue, correction de deux bugs d'affichage sur mobile.

1.4 — Les compétences, des fondations au framework

Les trois piliers (à maîtriser absolument)

Techno	Rôle	Analogie de la maison
HTML	La structure et le contenu	Les murs, les pièces, la charpente : ce qui existe sur la page.
CSS	Le style et la mise en page	La peinture, la déco, l'agencement : l'apparence.
JavaScript	Le comportement et l'interactivité	L'électricité et la domoterie : ce qui réagit et bouge.

À RETENIR

HTML + CSS + JavaScript sont le socle **incontournable** de tout frontend. Aucun framework ne les remplace : ils le complètent. On ne saute jamais cette étape.

Les compétences modernes attendues en entreprise

TypeScript · React · Vue.js · Next.js · Tailwind CSS · Sass · Vite · npm / pnpm · Tests (Jest, Playwright) · Git · Responsive / Mobile-first · Accessibilité (a11y)

- **TypeScript** : une version de JavaScript qui ajoute des « types » pour attraper les erreurs avant qu'elles n'arrivent. Devenu un standard en entreprise.
- **Un framework JavaScript** : **React** domine le marché, suivi de **Vue** et **Angular**. Ils permettent de construire des interfaces complexes à base de **composants** réutilisables. Conseil : en apprendre **un seul**, en profondeur.
- **Un méta-framework** comme **Next.js** (basé sur React) pour des sites rapides et bien référencés.
- **Le CSS moderne** : Flexbox, Grid, et un framework comme **Tailwind CSS** pour styliser efficacement.
- **Les outils du quotidien** : un gestionnaire de paquets (npm), un outil de build (Vite), et Git.

EXEMPLE CONCRET — UN COMPOSANT REACT

Voici un bouton « J'aime » qui compte les clics. C'est l'esprit du frontend moderne : de petits blocs autonomes (composants) qui gèrent leur propre état.

```
function BoutonJaime() {
  const [likes, setLikes] = useState(0);   // la mémoire du composant
```

```
  return (
    <button onClick={() => setLikes(likes + 1)}>
      ❤️ J'aime ({likes})
    </button>
  );
}
```

1.5 — Avec qui il travaille

Le Frontend est au carrefour de l'équipe : il reçoit les maquettes de l'**UX/UI Designer**, consomme les **API** livrées par le **Backend**, fait valider ses écrans par le **Product Owner**, et fait tester ses pages par le **QA**. Une bonne communication vaut autant que le code.

1.6 — Feuille de route du débutant (≈ 6 à 9 mois)

Étape 1 — Les fondations (1 à 2 mois)

HTML sémantique, CSS (Flexbox, Grid, responsive). Objectif : reproduire des pages statiques existantes.

Étape 2 — JavaScript (1 à 2 mois)

Variables, fonctions, tableaux, manipulation de la page (DOM), événements, et les appels réseau (fetch).

Étape 3 — Git & écosystème (continu)

Versionner son code, publier sur GitHub, comprendre npm et la ligne de commande.

Étape 4 — Un framework + TypeScript (2 à 3 mois)

Apprendre React en profondeur (composants, props, state, hooks), puis TypeScript.

Étape 5 — Qualité & mise en ligne (1 à 2 mois)

Tests, performance, accessibilité, déploiement sur Vercel ou Netlify.

PROJET POUR DÉBUTER

Construisez un **portfolio personnel** puis une **application météo** qui interroge une API publique. Ces deux projets couvrent à eux seuls 80 % des bases : mise en page, interactivité et connexion à une API.

PIÈGE CLASSIQUE DU DÉBUTANT

Sauter directement à React sans maîtriser HTML, CSS et JavaScript. Le framework devient alors une boîte noire incompréhensible. Patience : les fondations d'abord, le framework ensuite.

MODULE 2 — SPÉCIALISATION

Le Développeur Backend — les coulisses

Le cerveau invisible de l'application : logique métier, données, sécurité et performance. L'utilisateur ne le voit jamais, mais tout en dépend.

2.1 — En une phrase

Le **développeur Backend** construit la partie **cachée** de l'application, celle qui tourne sur le serveur : il reçoit les demandes, applique les règles, range et retrouve les données, vérifie les droits, et renvoie des réponses. Si le frontend est la salle du restaurant, le backend est la cuisine, le frigo et le coffre-fort réunis.

> **DÉFINITION — BACKEND**
>
> Le **backend** regroupe le code serveur, la **base de données** et la **logique métier** (les règles propres à l'entreprise : « un client ne peut pas commander un produit en rupture de stock »). L'utilisateur n'y a jamais accès directement ; il l'utilise via le frontend.

2.2 — Ses missions précises

- **Concevoir et développer des API** : le « menu » de services que le frontend (et les applis mobiles) viennent consommer. C'est le contrat entre les deux mondes.
- **Modéliser et gérer les données** : décider comment ranger les informations dans la base, écrire les requêtes pour les lire et les écrire efficacement.
- **Implémenter la logique métier** : les calculs, validations et règles propres à l'activité (prix, stocks, facturation, droits d'accès).
- **Gérer l'authentification et l'autorisation** : qui est connecté ? a-t-il le droit de faire cette action ? (mots de passe, jetons, sessions).
- **Assurer la sécurité** : protéger contre les attaques, ne jamais faire confiance aux données reçues, chiffrer les informations sensibles.
- **Garantir la performance et la montée en charge** : faire en sorte que tout reste rapide même avec des milliers d'utilisateurs (caches, optimisation des requêtes).
- **Intégrer des services externes** : paiement (Stripe), envoi d'e-mails, cartes, etc.
- **Écrire des tests et des journaux (logs)** pour vérifier que tout fonctionne et diagnostiquer les problèmes.

2.3 — La journée type

Matin

Daily, puis conception d'une nouvelle API « historique de commandes » : choix de l'adresse (route), des données renvoyées, des règles de sécurité.

Après-midi

Écriture de la requête en base de données, ajout de tests, optimisation d'une requête lente repérée en production, relecture de code d'un collègue.

2.4 — Les compétences clés

Les fondations transversales

- **Comprendre le web** : HTTP, le cycle requête/réponse, les codes de statut (200, 404, 500), DNS, et le modèle client-serveur.
- **Notions de système et de réseau** : ce qu'est un serveur, un port, un processus, et les bases de Linux et de la ligne de commande.
- **Git** : indispensable, comme pour tous les développeurs.

Choisir un langage et son framework

On apprend **un** langage backend en profondeur. Voici les plus courants en entreprise :

Langage	Framework typique	Réputé pour
JavaScript / TypeScript	Node.js + Express / NestJS	Cohérent avec le frontend, rapide à démarrer, énorme écosystème.
Python	Django / FastAPI	Lisible, idéal pour débuter, fort en data et en IA.
Java	Spring Boot	Robuste, très présent dans les grandes entreprises et la banque.
C#	.NET	Écosystème Microsoft, performant et structuré.
Go	standard library / Gin	Très rapide, parfait pour les services à grande échelle.
PHP	Laravel	Très répandu sur le web, encore largement utilisé.

Les bases de données

Le backend dialogue en permanence avec une base de données. Deux grandes familles :

Type	Exemples	Quand l'utiliser
Relationnelles (SQL)	PostgreSQL, MySQL	Données structurées et liées entre elles (clients, commandes, factures). Le choix par défaut.
NoSQL	MongoDB, Redis	Données souples, mise en cache ultra-rapide, gros volumes peu structurés.

EXEMPLE CONCRET — UNE ROUTE D'API

Voici, en Node.js avec Express, une mini-API qui renvoie un utilisateur. Le frontend appellera simplement l'adresse `/utilisateurs/42` pour obtenir ces données.

```
// Quand quelqu'un demande GET /utilisateurs/:id
app.get('/utilisateurs/:id', async (req, res) => {
  const user = await db.trouverUtilisateur(req.params.id); // lecture en base
  if (!user) return res.status(404).json({ erreur: 'Introuvable' });
  res.json(user); // réponse au format JSON
});
```

À RETENIR — LE FORMAT JSON

Frontend et backend s'échangent des données dans un format texte universel appelé **JSON** (du genre `{"nom": "Sophie", "age": 30}`). C'est la langue commune des API : compacte, lisible, comprise par tous les langages.

2.5 — Avec qui il travaille

Le Backend fournit les **API** au **Frontend** et aux **équipes Mobile** (le contrat doit être clair et stable). Il collabore avec le **DBA** pour les bases de données, avec le **DevOps** pour le déploiement, et avec l'**équipe Sécurité** pour les données sensibles.

2.6 — Feuille de route du débutant (≈ 7 à 10 mois)

Étape 1 — Bases de programmation (1 à 2 mois)

Un langage (Python ou JavaScript recommandé pour débuter), logique, structures de données.

Étape 2 — Comment fonctionne le web (1 mois)

HTTP, requêtes/réponses, JSON, le rôle d'un serveur. Construire une première mini-API.

Étape 3 — Bases de données (1 à 2 mois)

SQL avec PostgreSQL : créer des tables, lire, écrire, relier les données.

Étape 4 — Un framework backend (2 mois)

Express, FastAPI ou équivalent : routes, validation, authentification.

Étape 5 — Production (1 à 2 mois)

Tests, journaux, mise en cache (Redis), bases de Docker, déploiement sur Railway ou Render.

PROJET POUR DÉBUTER

Une **API de gestion de tâches (to-do)** avec base de données : créer, lire, modifier, supprimer (le fameux « CRUD »). Puis ajoutez l'authentification. Ce projet contient l'ADN de presque tout le backend.

MODULE 3 — SPÉCIALISATION

Le Développeur Full Stack — le pont

Le polyvalent qui touche aux deux mondes : frontend ET backend. Il comprend toute la chaîne, de l'écran de l'utilisateur jusqu'à la base de données.

3.1 — En une phrase

Le **développeur Full Stack** (« pile complète ») maîtrise à la fois le **frontend** et le **backend**. Il peut construire une fonctionnalité de bout en bout : l'écran que l'utilisateur voit, l'API qui le sert, et la base de données derrière. Il ne remplace pas les spécialistes ; il fait le lien et apporte une vision d'ensemble.

> **DÉFINITION — « STACK »**
>
> Une **stack** (pile) est l'ensemble des technologies empilées pour faire fonctionner une application : langage frontend + langage backend + base de données + outils. « Full stack » signifie « toute la pile ».

3.2 — Ses missions précises

- **Développer des fonctionnalités complètes** : de l'interface jusqu'aux données, sans dépendre d'un coéquipier pour chaque étape.
- **Concevoir le contrat d'API** : étant des deux côtés, il définit naturellement comment frontend et backend dialoguent.
- **Faire le pont entre les équipes** : traduire les besoins du frontend pour le backend et inversement.
- **Prototyper rapidement** : idéal pour transformer une idée en démonstration fonctionnelle (très recherché en startup).
- **Avoir des notions de déploiement** : savoir mettre son application en ligne (CI/CD de base, variables d'environnement, HTTPS).

> **MISE EN GARDE — « FULL STACK » N'EST PAS « EXPERT EN TOUT »**
>
> Personne ne maîtrise tout au même niveau. Un bon Full Stack est souvent **plus fort d'un côté** (front ou back) et **à l'aise de l'autre**. On parle parfois de profil « en T » : large sur l'ensemble, profond sur une spécialité.

3.3 — Le « 80/20 » réellement utile aujourd'hui

Plutôt que de tout apprendre superficiellement, un Full Stack efficace se concentre sur quelques compétences à fort rendement :

Compétence	Pourquoi elle compte le plus
Un framework frontend (React ou Vue)	Une seule, apprise en profondeur, suffit pour construire des interfaces réelles.
La conception d'API	C'est la compétence la plus universelle : tout dialogue entre composants passe par une API.
Une base de données (SQL)	Savoir modéliser et interroger les données reste central partout.
Le DevOps « juste assez »	Git, bases de CI/CD (GitHub Actions), variables d'environnement et secrets, comprendre DNS et HTTPS.
TypeScript	Une seule langue (JavaScript/TypeScript) des deux côtés simplifie énormément la vie.

À RETENIR

Le chemin le plus courant vers le Full Stack n'est pas d'apprendre les deux en même temps, mais de **maîtriser d'abord un côté** (front ou back), puis d'**élargir vers l'autre**. On devient Full Stack par extension, pas par dispersion.

3.4 — Feuille de route du débutant

Étape 1 — Choisir un point d'entrée

Commencer par le Frontend (plus visuel, plus motivant) ou le Backend (plus logique), selon votre goût. Suivre sa roadmap jusqu'au bout.

Étape 2 — Apprendre l'autre côté

Une fois un côté solide, apprendre l'essentiel de l'autre. Astuce : rester sur JavaScript/TypeScript des deux côtés (React + Node.js) pour ne pas changer de langage.

Étape 3 — Relier les deux

Construire une application complète : interface + API + base de données qui communiquent.

Étape 4 — Déployer en ligne

Mettre le tout en production et apprendre le minimum de DevOps pour y parvenir.

PROJET POUR DÉBUTER

Un **mini-réseau social** ou un **blog complet** : inscription/connexion, création de publications, affichage, commentaires. C'est le projet « full stack » par excellence, car il fait travailler les deux côtés et la base de données.

MODULE 4 — SPÉCIALISATION

L'Ingénieur DevOps — l'autoroute de livraison

Celui qui construit la route entre le code et les utilisateurs : automatiser, déployer, surveiller, et garder l'application en ligne, rapide et fiable.

4.1 — En une phrase

L'**ingénieur DevOps** automatise et fiabilise tout ce qui amène le code des développeurs jusqu'aux vrais utilisateurs. Il s'occupe des serveurs, des déploiements, de la surveillance et de la rapidité de livraison. Reprenons le restaurant : si les développeurs cuisinent, le DevOps gère la logistique, les livraisons, l'entretien des cuisines et fait en sorte que le service ne s'arrête jamais.

> **DÉFINITION — DEVOPS**
>
> **DevOps** est la contraction de *Development* (développement) et *Operations* (exploitation). C'est d'abord une **culture** et une **méthode de travail** qui rapproche ceux qui écrivent le code et ceux qui le font tourner, pour livrer plus souvent et plus sereinement. Ce n'est pas qu'un ensemble d'outils.

4.2 — Ses missions précises

- **Mettre en place des pipelines CI/CD** : automatiser le test, la construction et le déploiement du code à chaque modification.
- **Conteneuriser les applications** avec **Docker** : empaqueter une appli avec tout ce dont elle a besoin, pour qu'elle tourne à l'identique partout.
- **Orchestrer à grande échelle** avec **Kubernetes** : déployer, redémarrer et faire grandir automatiquement des centaines de conteneurs.
- **Gérer le cloud** (AWS, Azure, Google Cloud) : provisionner serveurs, réseaux et stockage.
- **Pratiquer l'Infrastructure as Code** (IaC) avec **Terraform** : décrire les serveurs sous forme de code, reproductible et versionné, plutôt que cliquer à la main.
- **Mettre en place l'observabilité** : journaux, métriques et alertes (Prometheus, Grafana) pour savoir, en direct, si quelque chose ne va pas.
- **Gérer les secrets et la sécurité du pipeline** : mots de passe, clés d'accès, analyses de vulnérabilités (DevSecOps).
- **Optimiser les coûts et la disponibilité** : que ça reste en ligne, rapide, et sans gaspiller d'argent.

4.3 — La boîte à outils du DevOps

Domaine	Outils emblématiques (2026)
Système	Linux, Bash (scripts), notions de réseau
Versionnement & CI/CD	Git, GitHub Actions, GitLab CI, Jenkins, ArgoCD (GitOps)
Conteneurs	Docker, puis Kubernetes (+ Helm)
Cloud	AWS, Microsoft Azure, Google Cloud
Infrastructure as Code	Terraform, Ansible
Observabilité	Prometheus, Grafana, ELK (Elasticsearch, Logstash, Kibana), Datadog
Scripting	Python, Go (très présent dans l'outillage cloud)

EXEMPLE CONCRET — UN PIPELINE CI/CD (IDÉE)

À chaque fois qu'un développeur envoie son code, une suite d'étapes automatiques se déclenche, sans intervention humaine :

```
# Déclenché à chaque envoi de code (push)
1. Récupérer le code
2. Lancer les tests automatiques
3. Construire l'image Docker de l'application
4. Déployer automatiquement en production
   # Si une étape échoue → tout s'arrête, l'équipe est alertée
```

À RETENIR — CI/CD

CI (Intégration Continue) = à chaque modification, on teste et on assemble automatiquement le code. **CD** (Déploiement/Livraison Continue) = on met en ligne automatiquement et fréquemment. Objectif : livrer souvent, vite, et sans stress.

DÉFINITION — CONTENEUR (DOCKER)

Un **conteneur** est une « boîte » qui contient l'application et tout son environnement (versions, dépendances). Avantage : elle tourne **exactement pareil** sur l'ordinateur du développeur, sur les serveurs de test et en production. Fini le fameux « pourtant ça marche chez moi ! ».

4.4 — DevOps, SRE, Platform Engineering : la nuance

Ces métiers très proches se côtoient dans les grandes entreprises. Pour un débutant :

- **DevOps** : la culture et les pratiques d'automatisation entre dev et exploitation.
- **SRE (Site Reliability Engineer)** : applique des méthodes d'ingénierie à la **fiabilité** (disponibilité, gestion des incidents, budgets d'erreur).

- **Platform Engineer** : construit la **plateforme interne** et les outils que les développeurs utilisent pour déployer en autonomie.

4.5 — Feuille de route du débutant (≈ 9 à 12 mois)

Étape 1 — Linux & réseau (1 à 2 mois)

Ligne de commande, permissions, scripts Bash, bases du réseau (DNS, IP, ports, load balancing).

Étape 2 — Git & un peu de code (1 mois)

Maîtriser Git, et savoir lire/écrire du Python pour automatiser.

Étape 3 — Docker (1 à 2 mois)

Images, conteneurs, volumes. Conteneuriser une application réelle.

Étape 4 — CI/CD (1 à 2 mois)

Construire un pipeline GitHub Actions qui teste et déploie automatiquement.

Étape 5 — Cloud & Kubernetes (2 à 3 mois)

Un cloud (AWS de préférence), puis Kubernetes une fois Docker bien compris.

Étape 6 — IaC & observabilité (1 à 2 mois)

Terraform pour décrire l'infrastructure, Prometheus + Grafana pour surveiller.

PROJET POUR DÉBUTER

Prenez n'importe quelle petite application, **conteneurisez-la avec Docker**, puis créez un **pipeline GitHub Actions** qui la teste et la déploie automatiquement à chaque modification. Vous aurez touché le cœur du DevOps.

PIÈGE CLASSIQUE

Se ruer sur Kubernetes sans solides bases de Linux, de réseau et de Docker. Résultat : on passe des heures bloqué sur des erreurs incompréhensibles. L'ordre compte : fondations d'abord, orchestration ensuite.

MODULE 5 — PANORAMA DES SPÉCIALISATIONS

Les autres métiers d'une grande équipe technique

Dans une grande entreprise, le développement ne se limite pas à quatre rôles. Voici les autres spécialistes, regroupés par pôle, chacun avec sa mission précise.

Chaque fiche suit la même logique : que fait-il, quelles tâches précises lui reviennent, et ses outils. L'idée n'est pas de tout apprendre, mais de comprendre **qui fait quoi** autour de vous.

5.1 — Construire le produit (au-delà du web)

📱 Développeur Mobile (iOS / Android)

Construit les applications installées sur les téléphones. C'est un cousin du frontend, mais avec ses propres règles (magasins d'applications, notifications, capteurs, mode hors-ligne).

Tâches précises : développer l'interface mobile, gérer la performance et la batterie, publier sur l'App Store / Google Play, s'adapter à des centaines de modèles d'écrans.

Swift (iOS) · Kotlin (Android) · React Native · Flutter

🎨 UX / UI Designer

Conçoit l'**expérience** (UX : le parcours, la logique, la facilité d'usage) et l'**interface** (UI : l'apparence, les couleurs, la typographie). Il livre les **maquettes** que le frontend intègre ensuite.

Tâches précises : recherche utilisateur, schémas de parcours (wireframes), maquettes haute-fidélité, système de design (composants réutilisables), tests d'utilisabilité.

Figma · Design system · Prototypage · Accessibilité

5.2 — Garantir la qualité

✅ QA Engineer / Testeur

Garant que le logiciel fonctionne comme prévu **avant** qu'il n'arrive chez l'utilisateur. Il cherche les bugs de manière méthodique, là où les développeurs n'ont pas pensé.

Tâches précises : écrire des scénarios de test, tester manuellement les nouvelles fonctionnalités, **automatiser les tests** qui se relancent à chaque modification, rédiger des rapports de bugs clairs, valider avant chaque mise en production.

Tests manuels | Playwright / Cypress | Selenium | Tests de charge

5.3 — Maîtriser la donnée et l'intelligence

🔧 Data Engineer

Construit la « tuyauterie » de la donnée : il collecte, nettoie et achemine d'énormes volumes d'informations vers les endroits où on les analyse. Sans lui, les données restent inexploitables.

Tâches précises : créer des pipelines de données (ETL), alimenter des entrepôts de données (data warehouse), garantir la fiabilité et la fraîcheur des données.

Python / SQL | Spark | Airflow | Snowflake / BigQuery

📊 Data Scientist / Data Analyst

Donne du sens aux données. L'**Analyst** répond aux questions de l'entreprise avec des tableaux de bord et des rapports ; le **Scientist** va plus loin avec des modèles statistiques et de prédiction.

Tâches précises : explorer les données, créer des visualisations et tableaux de bord, bâtir des modèles prédictifs, présenter des recommandations aux décideurs.

Python (pandas) | SQL | Statistiques | Power BI / Tableau

🤖 ML / AI Engineer

Met l'intelligence artificielle en production : il transforme les modèles d'apprentissage automatique en services réels, fiables et rapides, utilisables par l'application.

Tâches précises : entraîner et affiner des modèles, les déployer sous forme d'API, surveiller leur performance dans le temps (MLOps), intégrer des modèles d'IA générative.

Python · PyTorch / TensorFlow · MLOps · API d'IA

5.4 — Faire tourner et protéger la plateforme

☁️ Cloud / Platform Engineer

Construit et entretient l'infrastructure cloud et la plateforme interne sur laquelle tournent toutes les applications. Très proche du DevOps, avec un focus sur l'**outillage en libre-service** pour les développeurs.

Tâches précises : architecturer l'infrastructure cloud, créer des modèles réutilisables, maîtriser les coûts (FinOps), offrir aux équipes des moyens de déployer en autonomie.

AWS / Azure / GCP · Terraform · Kubernetes

⚙️ SRE — Site Reliability Engineer

Veille à ce que les services restent **fiables et disponibles**, même sous forte charge. Il applique une démarche d'ingénieur à la stabilité et gère les incidents quand tout s'enflamme.

Tâches précises : définir des objectifs de disponibilité (SLO), automatiser la réponse aux pannes, mener les analyses post-incident, réduire la « charge manuelle ».

Observabilité · Gestion d'incidents · Automatisation

🗄️ DBA — Administrateur de Base de Données

Le gardien des bases de données : il garantit qu'elles sont rapides, sûres, sauvegardées et qu'aucune donnée n'est jamais perdue. Crucial dès que les volumes deviennent énormes.

Tâches précises : optimiser les requêtes lentes, gérer les sauvegardes et la restauration, sécuriser les accès, planifier la montée en charge.

PostgreSQL / Oracle · Optimisation SQL · Sauvegarde / restauration

🔒 Ingénieur Sécurité / DevSecOps

Protège l'entreprise et ses utilisateurs contre les attaques et les fuites de données. Le DevSecOps intègre la sécurité **directement dans le pipeline**, au lieu d'en faire une vérification de dernière minute.

Tâches précises : tests d'intrusion, analyse de vulnérabilités automatisée, gestion des secrets (Vault), modèles « zéro confiance », sensibilisation des équipes.

SAST / DAST | HashiCorp Vault | Zero-trust | Conformité

5.5 — Encadrer, décider et piloter

Rôle	Sa mission en quelques mots
Tech Lead	Développeur expérimenté, référent technique de son équipe : il guide les choix, relit le code clé et débloque ses coéquipiers, tout en continuant souvent à coder.
Architecte logiciel	Conçoit la structure d'ensemble des systèmes : quelles briques, comment elles communiquent, comment elles tiendront la charge dans 3 ans. Vision long terme.
Engineering Manager	Encadre les développeurs côté humain : carrières, recrutement, organisation, bien-être de l'équipe. Moins de code, plus de management.
CTO (Directeur Technique)	Porte la vision et la stratégie technique de toute l'entreprise. Fait le lien entre la technique et la direction.
Product Manager / Product Owner	Définit le **quoi** et le **pourquoi** : quelles fonctionnalités construire, pour quels utilisateurs, dans quel ordre de priorité. Le « capitaine » du produit.
Scrum Master	Veille à la fluidité de l'équipe : il anime les rituels Agile, lève les obstacles et protège l'équipe des distractions. Un facilitateur, pas un chef.

À RETENIR

Plus on monte dans cette liste, moins on écrit de code et plus on prend de décisions et on coordonne. Beaucoup de développeurs choisissent à un moment entre deux chemins de carrière : la voie **technique experte** (Tech Lead → Architecte) ou la voie **management** (Engineering Manager → Directeur).

MODULE 6 — LA MACHINE EN MOUVEMENT

Comment tout le monde travaille ensemble

Connaître les rôles, c'est bien. Comprendre comment ils s'enchaînent pour transformer une idée en logiciel utilisé par des millions de personnes, c'est mieux.

6.1 — Le cycle de vie d'un logiciel

Tout produit suit, en boucle, les mêmes grandes étapes. Chaque spécialiste intervient au bon moment :

1. **Idée & besoin**
   Le **Product Manager** identifie un besoin utilisateur et décide quoi construire.
2. **Conception**
   L'**UX/UI Designer** dessine les maquettes ; l'**Architecte** et le **Tech Lead** décident comment le construire.
3. **Développement**
   Les développeurs **Frontend**, **Backend**, **Full Stack** et **Mobile** écrivent le code.
4. **Test**
   Le **QA** vérifie que tout fonctionne et ne casse rien d'existant.
5. **Déploiement**
   Le **DevOps** met le code en production via le pipeline automatisé.
6. **Surveillance & maintenance**
   Le **SRE** et le **DevOps** surveillent ; les bugs et retours nourrissent… la prochaine idée. La boucle recommence.

6.2 — La méthode Agile et Scrum

Les grandes entreprises ne construisent pas un logiciel d'un seul bloc pendant deux ans. Elles avancent par **petits cycles courts et répétés**. C'est la philosophie **Agile**, dont **Scrum** est la mise en pratique la plus répandue.

> **DÉFINITION — LE « SPRINT »**
>
> Un **sprint** est un cycle de travail court (souvent 2 semaines) au bout duquel l'équipe livre quelque chose de concret et fonctionnel. On préfère livrer un petit morceau utile toutes les 2 semaines plutôt qu'un gros bloc dans un an.

Rituel Scrum	À quoi il sert
Planification de sprint	L'équipe choisit ce qu'elle s'engage à réaliser pendant les 2 prochaines semaines.
Mêlée quotidienne (daily)	15 minutes debout, chaque matin : ce que j'ai fait, ce que je fais, mes blocages.

Revue de sprint	Démonstration du travail réalisé aux parties prenantes.
Rétrospective	L'équipe réfléchit : qu'est-ce qui a bien marché, qu'améliorer la prochaine fois ?

6.3 — Le voyage concret d'une fonctionnalité

Suivons une demande réelle — « Ajouter un bouton J'aime sur les publications » — du début à la fin, pour voir chaque rôle s'activer.

Qui	Fait quoi
Product Manager	Écrit la demande : « En tant qu'utilisateur, je veux aimer une publication pour montrer mon intérêt. »
UX/UI Designer	Dessine le bouton, son apparence aimée / non aimée, l'emplacement et l'animation.
Backend	Crée une API pour enregistrer un « j'aime » et une table en base de données pour le stocker.
Frontend	Intègre le bouton, le relie à l'API, affiche le nombre de « j'aime » en temps réel.
QA	Teste : cliquer, recliquer, sur mobile, hors-ligne, avec un compte non connecté.
DevOps	Le pipeline teste, construit et déploie automatiquement la nouvelle version en production.
SRE / DevOps	Surveille que le nouveau code n'a pas ralenti l'application ni provoqué d'erreurs.

LE VOYEZ-VOUS ?

Une fonctionnalité aussi simple qu'un bouton « J'aime » mobilise **sept spécialités** différentes. C'est exactement cela, l'organisation d'une grande entreprise : chaque tâche précise revient à son spécialiste, et la qualité naît de leur coordination.

6.4 — Git : le langage commun de la collaboration

Tous les développeurs, quel que soit leur métier, partagent un outil : **Git**. Il permet à des dizaines de personnes de travailler sur le même code sans s'écraser les uns les autres.

DÉFINITION — GIT ET LE FLUX DE TRAVAIL

Git enregistre l'historique complet du code. On travaille sur une **branche** séparée (sa propre copie), puis on propose ses changements via une **pull request** que les collègues **relisent** avant de **fusionner** dans le code principal. Rien n'arrive en production sans avoir été relu.

Action	En français
`commit`	Enregistrer une étape de travail avec un message décrivant le changement.
`branch`	Créer une ligne de travail parallèle, sans toucher au code principal.
`pull request`	Proposer ses changements à l'équipe pour relecture avant intégration.
`merge`	Fusionner le travail validé dans le code principal.

`code review`	Relecture du code d'un collègue : garde-fou qualité essentiel.

À RETENIR

Git n'est pas optionnel. C'est le **premier outil** à apprendre, quel que soit le métier visé (frontend, backend, DevOps…). C'est la base de tout travail d'équipe en informatique.

MODULE 7 — PASSER À L'ACTION

Par où commencer ? Votre feuille de route

Vous connaissez maintenant tous les métiers. Reste la question la plus importante : par où, vous, commencez-vous ?

7.1 — Choisir sa voie selon ce qui vous motive

Si vous aimez…	Orientez-vous vers…
Le visuel, le design, voir un résultat immédiat à l'écran	**Frontend**
La logique, les énigmes, les données et l'architecture invisible	**Backend**
Tout toucher et construire des choses de bout en bout	**Full Stack**
L'automatisation, les systèmes, faire tourner des machines à grande échelle	**DevOps / Cloud**
Les chiffres, les statistiques, donner du sens aux données	**Data / IA**
Chasser les défauts, la rigueur et la méthode	**QA**

CONSEIL POUR LES INDÉCIS

Commencez par le **Frontend**. C'est l'entrée la plus accessible : on voit vite un résultat concret, ce qui maintient la motivation. Vous pourrez toujours bifurquer ensuite ; les fondations (Git, JavaScript, HTTP) servent partout.

7.2 — Les fondations communes à TOUS les métiers

Quelle que soit la voie, ces bases reviennent partout. Investissez-y dès le début :

- **Comment fonctionne Internet** : client/serveur, HTTP, DNS, le voyage d'une requête (Module 0).
- **Au moins un langage de programmation**, appris en profondeur plutôt que cinq en surface.
- **Git et la ligne de commande** : non négociables.
- **Savoir lire une documentation** et résoudre un problème par soi-même (la compétence la plus sous-estimée).

7.3 — Une méthode d'apprentissage qui marche

Apprendre un concept, puis l'appliquer aussitôt

Ne cumulez pas les tutoriels sans coder. Après chaque notion, construisez quelque chose, même minuscule.

Construire des projets concrets

Un portfolio, une to-do list, une appli météo, un blog. Les projets valent dix fois les cours passifs et remplissent votre portfolio.

Publier son code sur GitHub

C'est votre vitrine pour les recruteurs et votre carnet de progression.

Lire le code des autres

Les projets open source sont une mine d'or pour voir comment les pros structurent leur travail.

Être régulier plutôt qu'intense

Une heure par jour bat dix heures le dimanche. La constance crée la maîtrise.

7.4 — Les erreurs classiques à éviter

LES PIÈGES LES PLUS FRÉQUENTS

- **Le « tutorial hell »** : enchaîner les tutoriels sans jamais construire seul. Sortez-en en codant vos propres projets.
- **Vouloir tout apprendre en même temps** : choisissez une voie, une techno, et allez au bout.
- **Sauter les fondations** pour aller droit aux frameworks à la mode. Les bases d'abord.
- **Se comparer aux experts** : ils ont commencé exactement où vous êtes. Comparez-vous à vous-même d'il y a un mois.
- **Négliger la communication** : en entreprise, savoir expliquer et collaborer compte autant que coder.

LE MOT DE LA FIN

Aucun de ces métiers ne s'apprend en une nuit, et aucun ne s'apprend par la lecture seule. Choisissez une voie qui vous attire, maîtrisez les fondations, construisez des projets réels, et soyez patient avec vous-même. Chaque expert que vous admirez est simplement un débutant qui n'a pas abandonné. Bienvenue dans le développement logiciel.

ANNEXES

Références rapides

À garder sous la main : un glossaire, un tableau récapitulatif de tous les rôles, et un comparatif des langages pour choisir le vôtre.

A.1 — Tableau récapitulatif de tous les rôles

Rôle	Mission centrale	Compétences clés
Frontend	L'interface visible	HTML, CSS, JS/TS, React
Backend	La logique et les données serveur	Python/Node/Java, SQL, API
Full Stack	Les deux côtés de bout en bout	Front + Back + API
DevOps	Automatiser livraison et exploitation	Docker, CI/CD, Cloud, Terraform
Mobile	Les applications sur téléphone	Swift, Kotlin, Flutter
QA	Garantir l'absence de bugs	Tests manuels & automatisés
Data Engineer	La tuyauterie de la donnée	Python, SQL, Spark, Airflow
Data Scientist / Analyst	Donner du sens aux données	Statistiques, Python, BI
ML / AI Engineer	Mettre l'IA en production	PyTorch, MLOps, API d'IA
Cloud / Platform	Infrastructure et plateforme interne	AWS/Azure/GCP, Terraform
SRE	Fiabilité et disponibilité	Observabilité, incidents
DBA	Santé des bases de données	SQL avancé, sauvegardes
Sécurité / DevSecOps	Protéger données et systèmes	Pentest, Vault, conformité
UX/UI Designer	Expérience et apparence	Figma, design system
Tech Lead / Architecte	Guider les choix techniques	Expérience + vision système
PM / PO / Scrum Master	Piloter le produit et l'équipe	Priorisation, Agile

A.2 — Quel langage pour débuter ?

Langage	Idéal pour	Pourquoi débuter avec lui
JavaScript / TypeScript	Frontend, Full Stack, Backend (Node)	Le seul langage qui couvre front ET back. Indispensable pour le web.
Python	Backend, Data, IA, DevOps	Le plus lisible pour un débutant ; ouvre le plus de portes.
Java / C#	Backend en grande entreprise	Très demandés, structurés, robustes (banque, assurance, ERP).

Go	DevOps, services à grande échelle	Rapide et moderne ; précieux pour l'infrastructure cloud.
SQL	Tous les métiers de la donnée	Ce n'est pas un langage « complet » mais il est incontournable partout.

LE CONSEIL LE PLUS UTILE

Si vous hésitez : **JavaScript/TypeScript** si vous visez le web (front, full stack), **Python** si vous visez le backend, la data ou l'IA. Les deux sont d'excellents premiers langages. L'important n'est pas *lequel*, mais d'en choisir un et de s'y tenir.

A.3 — Glossaire complet

Terme	Définition simple
Agile	Façon de travailler par petits cycles courts et itératifs plutôt qu'en un seul grand bloc.
API	Contrat et menu de services qu'un programme expose pour que d'autres l'utilisent.
Backend	La partie cachée qui tourne sur le serveur : logique et données.
Base de données	Le grand classeur organisé où l'on range durablement les informations.
Bug	Comportement non voulu du logiciel ; une erreur à corriger.
CI/CD	Intégration et déploiement continus : tester et livrer le code automatiquement.
Cloud	Serveurs et services loués à distance (AWS, Azure, Google Cloud) au lieu de posséder ses machines.
Conteneur / Docker	Boîte qui empaquette une appli et son environnement pour qu'elle tourne pareil partout.
CRUD	Les 4 opérations de base sur des données : Créer, Lire, Modifier, Supprimer.
Déploiement	Action de publier le logiciel pour les vrais utilisateurs.
DevOps	Culture et pratiques rapprochant développement et exploitation pour livrer vite et bien.
DNS	L'annuaire d'Internet qui traduit un nom de site en adresse numérique (IP).
Framework	Boîte à outils structurée qui évite de tout réécrire (React, Django…).
Frontend	La partie visible dans le navigateur : interface et interactivité.
Git	Outil qui sauvegarde l'historique du code et permet le travail à plusieurs.
HTTP(S)	Le protocole de dialogue entre navigateur et serveur (le « S » = sécurisé).
IaC	Infrastructure as Code : décrire ses serveurs sous forme de code reproductible.
JSON	Format texte universel d'échange de données entre programmes.
Kubernetes	Chef d'orchestre qui gère automatiquement des centaines de conteneurs.
Open source	Logiciel dont le code est public, librement consultable et réutilisable.
Production (« prod »)	L'environnement réel utilisé par les vrais clients.

Pull request	Proposition de changements de code soumise à la relecture de l'équipe.
Responsive	Interface qui s'adapte à toutes les tailles d'écran (mobile, ordinateur).
Serveur	Ordinateur allumé en permanence qui « sert » des données aux clients.
SQL	Langage pour interroger et manipuler les bases de données relationnelles.
Sprint	Cycle de travail court (souvent 2 semaines) en méthode Scrum.
Stack	Ensemble des technologies empilées pour faire tourner une application.
TypeScript	Version de JavaScript ajoutant des types pour éviter des erreurs.

Fin du guide — **Les Métiers du Développement Logiciel** · Édition 2026

Un manuel pédagogique pour comprendre chaque spécialisation et choisir votre voie.