• ÉDITION DÉFINITIVE — GUIDE DE RÉFÉRENCE 2026

Maîtriser Lovable de A à Z

Le guide complet pour créer des applications web avec l'IA — méthodologie, prompts, astuces, checklists et exemples concrets pour les non-développeurs et les professionnels.

🚀 Mise en production
💡 51 astuces pratiques
📋 Checklists actionnables
🔧 Prompts prêts à l'emploi
🗄️ Supabase & GitHub

6	51	15+	10
phases méthodologiques	astuces & conseils	templates de prompts	checklists actionnables

📋 SOMMAIRE

Qu'est-ce que Lovable ?

Lovable est un constructeur d'applications web piloté par l'IA. Vous décrivez ce que vous voulez construire en langage naturel — et l'outil génère le code correspondant en temps réel.

Lovable n'est ni un générateur de sites vitrines classique, ni un outil drag-and-drop, ni un IDE traditionnel. C'est une **plateforme de développement IA complète** qui produit des applications React + Tailwind CSS connectables à Supabase, GitHub, Stripe et de nombreuses APIs.

> 💡 **La bonne définition**
>
> Ce que Lovable crée, c'est une vraie application web avec interface utilisateur, logique, flux d'authentification, bases de données, fonctions backend, options de publication et synchronisation GitHub — pas une simple maquette.

Ce que Lovable fait très bien

Lovable excelle sur les projets combinant pages, formulaires, tableaux, filtres, graphiques, authentification et logique CRUD :

- Tableaux de bord et portails clients
- Outils internes d'entreprise
- Interfaces SaaS et MVP rapides
- Marketplaces légères et processus de réservation
- Panels d'administration
- Landing pages et microsites de campagne

Où Lovable montre ses limites

- Règles métier hautement personnalisées et logique complexe
- Architectures atypiques et gestion d'état élaborée
- Gestion intensive des exceptions dans des workflows interconnectés
- Applications nécessitant des tests unitaires stricts dès le départ

> ⚠️ **Le bon modèle mental**
>
> "Lovable peut *accélérer* le développement d'une large gamme d'applications web modernes — ce n'est pas un substitut à toute la chaîne de développement, mais un accélérateur puissant en début de projet."

La stack technique générée

Couche	Technologie	Notes
Frontend	**React 18 + TypeScript**	Composants modernes avec hooks
Style	**Tailwind CSS + ShadCN/UI**	Design system cohérent et responsive
Backend	**Supabase**	PostgreSQL + Auth + Storage + Edge Functions
Versionnement	**GitHub**	Sync automatique à chaque modification
Déploiement	**Lovable / Netlify / Vercel / Cloudflare**	Au choix selon les besoins

02 — COMPARATIF

Lovable vs les autres outils

Chaque outil a sa place dans l'écosystème. Voici comment choisir intelligemment.

Lovable

Builder IA no-code

Code requis	Aucun
Vitesse MVP	Très rapide
Backend	Supabase intégré
Contrôle code	Partiel
Prix	Crédits

Claude Code

IDE IA terminal

Code requis	Bases utiles
Vitesse MVP	Moyenne
Backend	Tout backend
Contrôle code	Total
Prix	Abonnement fixe

Cursor

IDE IA code-first

Code requis	Oui
Vitesse MVP	Bonne
Backend	Tout backend
Contrôle code	Total
Prix	Abonnement fixe

Bubble

No-code traditionnel

Code requis	Aucun
Vitesse MVP	Lente
Backend	Propriétaire
Contrôle code	Limité
Prix	Élevé

Quand choisir quoi ?

Situation	Outil recommandé	Pourquoi
MVP à valider rapidement	**Lovable**	Vitesse maximale, preview en temps réel
MVP + logique complexe	**Lovable + Claude Code**	Lovable pour le frontend, Claude pour la logique
Projet développeur solo	**Cursor + Claude**	Contrôle total du code, pas de limite de crédits
Site vitrine / marketing	**Webflow / WordPress**	Outils dédiés, meilleur SEO natif
App complexe en production	**Lovable puis migration GitHub**	Démarrer vite, puis prendre le contrôle

🎯 **Le conseil des experts**

La combinaison

Lovable + Claude Code via GitHub

est le standard dans la communauté des builders IA sérieux. Lovable pour aller vite, Claude pour aller loin. La communauté rapporte des réductions de coûts Lovable pouvant atteindre 90 % grâce à ce combo.

Avant de commencer : les fondamentaux

La plus grande erreur est de se précipiter dans Lovable avec une idée vague. 20 minutes de préparation évitent des heures de corrections.

Comprendre le système de crédits

Lovable fonctionne avec un système de crédits — chaque génération de code en consomme. Le Plan Mode (chat, planification, débogage) ne consomme **pas** de crédits.

Forfait	Crédits/mois	Usage recommandé
Gratuit	5 crédits	Test et découverte uniquement
Starter (~20$/mois)	100 crédits	1 à 2 projets MVP
Pro (~50$/mois)	Crédits étendus	Projets sérieux, itérations fréquentes
Teams	Crédits partagés	Équipes et collaboration

💰 **Économisez vos crédits**

Utilisez le
Plan Mode 60 à 70 % du temps
pour déboguer, planifier et itérer sans consommer un seul crédit. Ne cliquez « Implémenter » que quand vous êtes sûr du résultat.

Le brief produit : les 4 questions fondamentales

📝 Brief produit — Checklist de départ

- [ ] **Qu'est-ce que je construis ?** — Décrivez le produit en 1 à 2 phrases sans jargon technique
- [ ] **Pour qui ?** — Définissez les personas et rôles utilisateurs (Admin, Utilisateur, Visiteur…)
- [ ] **Pourquoi ?** — Quel problème cela résout-il ? Quelle valeur apporte-t-il ?
- [ ] **Quelle action unique principale ?** — Une seule action que l'app doit rendre fluide
- [ ] **Niveau de maturité ?** — POC, MVP exploitable, ou production complète ?
- [ ] **Quelles exclusions ?** — Ce que l'app NE fera PAS dans cette version
- [ ] **Données** — Avez-vous des CSV, APIs ou données à intégrer ? Si oui, documenter chaque colonne

Configuration initiale : ce qu'il faut faire en premier

1. **Créer un compte Lovable** — Choisir le bon forfait selon votre usage
2. **Connecter GitHub immédiatement** — Dès le premier MVP, activez la synchronisation. Chaque modification devient un commit automatique
3. **Préparer votre Knowledge File** — Le document de référence de votre projet (voir section 6)
4. **Rédiger le Prompt 0** — Le contexte immuable du projet, avant même de toucher à l'interface

---

La méthode en 6 phases séquentielles

La règle fondamentale : **une phase doit être validée explicitement avant de passer à la suivante.** Cette discipline évite 80 % des problèmes classiques.

🚨 **L'erreur la plus fréquente**

Commencer par « Crée-moi une belle app avec un tableau de bord moderne. » Ce type de demande pousse Lovable à inventer des filtres, des données fictives, des règles métier et des pages non demandées.

PHASE 0

🎯 Contexte immuable

Définir le cadre stratégique : objectif produit, public cible, contraintes fortes, exclusions explicites, niveau de maturité (POC / MVP / Production). À ce stade, on ne parle pas d'interface, de code ou de données.

Optionnel mais recommandé · Stratégie · 0 code

PHASE 1

🏗️ Architecture & périmètre fonctionnel

Définir la structure globale : pages/vues principales, modules fonctionnels, flux entre modules, inclusions et exclusions. Pas encore de design, de données ou de logique métier détaillée.

Structure · Navigation · Flux utilisateur

PHASE 2

🗃️ Données & sources

Sécuriser la base logique : sources de données, colonnes exactes avec types et contraintes de nullité, fréquence de mise à jour, règles contractuelles. Un CSV n'est jamais « juste importé » — il doit être décrit colonne par colonne.

Critique · Colonnes contractuelles · No mock data

PHASE 3

⚙️ Logique métier & règles fonctionnelles

Empêcher Lovable d'inventer des comportements. Chaque règle métier doit être écrite explicitement : règles de calcul, exceptions, priorités, cas limites et règles d'erreur. Si une règle n'est pas écrite, elle n'existe pas.

Règles explicites · Pas d'inférence · Cas limites

PHASE 4

🎨 UI/UX & parcours utilisateur

Construire l'interface uniquement après avoir verrouillé l'architecture, les données et la logique. Définir les écrans, les états (chargement / erreur / vide), les parcours et l'accessibilité. L'UI affiche et déclenche des actions — elle ne doit pas contenir la logique métier principale.

Mobile-first · États vides · Accessibilité

PHASE
5

✅ Vérifications, tests & non-régression

Le dernier prompt ne sert pas à ajouter des fonctionnalités — il vérifie que Lovable n'a pas modifié le périmètre, renommé des colonnes, inventé des données, déplacé la logique dans l'UI, ajouté des filtres non demandés ou cassé une fonctionnalité existante.

Non-régression | Rapport de conformité | Dernier check

Les 10 règles d'or

🏆 Règles d'or Lovable

- → Ne jamais demander une application complète en un seul prompt
- → Ne jamais commencer par l'interface utilisateur
- → Ne jamais importer un CSV sans description contractuelle colonne par colonne
- → Ne jamais laisser Lovable déduire une règle métier
- → Ne jamais accepter de données fictives silencieuses
- → Ne jamais passer à l'étape suivante sans validation explicite
- → Toujours rappeler les contraintes validées dans chaque nouveau prompt
- → Toujours définir les exclusions aussi précisément que les inclusions
- → Toujours prévoir les états d'erreur, de chargement et les cas vides
- → Toujours terminer par un prompt de tests et non-régression

Structure type d'un prompt fiable

TEMPLATE DE PROMPT STRUCTURÉ — À copier-coller

```
Étape X — [Nom de l'étape]

Objectif :
[Objectif précis et mesurable de cette étape]

Contexte déjà validé :
[Rappel des contraintes et décisions des étapes précédentes]

Instructions :
1. [Instruction précise]
2. [Instruction précise]

Interdictions :
1. Ne pas renommer les colonnes contractuelles
2. Ne pas ajouter de fonctionnalités hors périmètre
3. Ne pas modifier les pages X et Y

Critères de validation :
1. [Ce qui doit être vrai pour valider l'étape]

→ Fin : "Valides-tu ce prompt pour passer à l'étape suivante ?"
```

L'art du prompting : 4 niveaux de maîtrise

La qualité de votre prompt détermine directement la qualité du résultat. Un prompt de 50 mots bien structuré bat un pavé de 500 mots confus.

Niveau 1

🎮 Training-Wheels

Débutant, ateliers, premier jet

Structure Markdown avec titres clairs : Contexte → Tâche → Contraintes. Idéal pour apprendre la rigueur du prompting.

```
# Contexte
Générer un CRUD contacts.

## Tâche
Création tables + UI.

### Contraintes
Pas de CSS custom.
```

Niveau 2

🚴 Sans les roulettes

Prompt court, utilisateur pressé

Même logique mais les titres disparaissent. On garde l'ordre Contexte-Tâche-Contraintes dans un seul paragraphe.

```
"Génère un CRUD contacts
(Postgres). UI Tailwind,
aucun refactor sans
validation explicite."
```

Niveau 3

🤖 Méta-prompting

Gagner du temps, découvrir la structure

Demandez à un LLM d'écrire le prompt pour vous en lui donnant juste le sujet. L'IA structure la demande à votre place.

```
"Tu es prompt-engineer.
Rédige le meilleur prompt
pour créer un CRM simple
avec Supabase en précisant
tout le contexte possible."
```

Niveau 4

🔄 Reverse prompting

Documentation, clonage, REX d'équipe

En fin de session, l'IA résume tout : le prompt + les erreurs évitées = base parfaite pour la V2 ou pour partager.

```
"Résume ce projet, liste
les pièges rencontrés et
donne un prompt prêt à
l'emploi pour un clone."
```

Techniques avancées de prompting

🔒 Verrouiller les fichiers et composants

PROMPT DE VERROUILLAGE

```
Implement modifications to the feature while ensuring core
functionality, other features, and processes remain unaffected.
Evaluate dependencies and identify potential risks before moving
forward. Conduct thorough testing to verify there are no
regressions. Exercise caution — pause if uncertain.

// Ou en français :
Ne modifie pas /pages/Settings.tsx ni /shared/Layout.tsx.
Concentre les modifications uniquement sur /components/Dashboard.tsx.
```

💬 Demander les clarifications avant de coder

CLARIFICATION PRÉALABLE

```
Ask me any questions you need in order to fully understand
what I want from this feature and how I envision it.
Do not write any code until I have answered your questions.
```

Comparaison : mauvais vs bons prompts

❌ MAUVAIS

"Affiche les meilleurs clients."

✅ BON

"Un meilleur client est un client dont le CA total est supérieur à 10 000 € sur les 12 derniers mois. Les commandes annulées sont exclues."

❌ MAUVAIS

"Utilise mon CSV de clients."

✅ BON

"Le CSV contient : client_id (string, non-null), nom (string), email (nullable), statut (enum: actif/inactif/prospect). Ces noms sont contractuels."

Checklist anti-blocage

🛡️ À vérifier avant chaque génération

- [ ] Input ET output attendus clairement définis dans le prompt
- [ ] Stack technique nommée : React, Tailwind, Supabase Edge Functions
- [ ] Règle anti-refactor incluse : « Pas de refactor sans validation explicite »
- [ ] Plan Mode activé et utilisé pour les clarifications avant la génération
- [ ] Un seul sujet par prompt (pas de multi-sujet)
- [ ] Rôle utilisateur précisé si l'app a plusieurs profils
- [ ] Reverse prompt généré et archivé après chaque session importante
- [ ] Version épinglée avant toute modification importante

06 — KNOWLEDGE FILE

Le Knowledge File : la mémoire de votre projet

Le Knowledge File est le cerveau de votre projet. Il est envoyé avec chaque prompt et donne à Lovable le contexte complet pour éviter les incohérences.

> 🧠 **C'est votre mini-PRD vivant**
>
> Le Knowledge File contient la vision produit, les personas, les parcours, les fonctionnalités clés, les guidelines UI et les règles par rôle. Mettez-le à jour après chaque phase majeure.

Template complet de Knowledge File

📄 knowledge.md — Template à personnaliser

```
1. VISION PRODUIT

Nom : [Nom de l'application]
Objectif : [Ce que fait l'app en 1 phrase]
Problème résolu : [Le problème utilisateur précis]
Niveau : [POC / MVP / Production]

2. UTILISATEURS ET RÔLES

Rôle Admin :
  - Permissions : [liste complète]
  - Pages accessibles : [liste]

Rôle Utilisateur :
  - Permissions : [liste]
  - Pages accessibles : [liste]

Rôle Visiteur : landing page uniquement

3. FONCTIONNALITÉS MVP

✅ [Fonctionnalité 1] — [Description courte]
✅ [Fonctionnalité 2] — [Description courte]

❌ EXCLUSIONS (ne pas implémenter) :
- [Exclusion 1] / [Exclusion 2]

4. STACK TECHNIQUE

Frontend : React 18 + TypeScript
Styling : Tailwind CSS + ShadCN/UI
Backend : Supabase (PostgreSQL + Auth + Edge Functions)
Versionning : GitHub

5. RÈGLES LOVABLE

- Ne jamais refactoriser sans validation explicite
- Ne jamais renommer les colonnes de données
- Ne jamais ajouter de fonctionnalités hors périmètre
- Ne jamais utiliser de données fictives ou mockées
- Toujours préciser l'état vide, l'état erreur, l'état chargement
```

> ⚡ **Générer le Knowledge File automatiquement**
>
> Demandez au Plan Mode : *« Génère le knowledge file de mon projet basé sur les fonctionnalités déjà implémentées. »* Mettez-le à jour régulièrement après chaque phase majeure.

07 — GITHUB

GitHub : versionnement et collaboration

GitHub est votre filet de sécurité. Connectez-le dès le premier MVP — chaque modification Lovable devient un commit automatique.

Connexion GitHub — Étape par étape

1. Dans Lovable, cliquez sur le bouton **GitHub** en haut à droite
2. Autorisez l'accès à votre compte GitHub
3. Créez un nouveau repository ou sélectionnez un existant
4. Dès lors, chaque modification génère un commit automatique
5. Vous pouvez cloner le repo localement ou l'ouvrir dans Claude Code / Cursor

Stratégie de branches recommandée

Branche	Usage	Règle
`main`	Production stable	Ne jamais pousser directement depuis Lovable en prod
`dev`	Développement actif	Branche de travail Lovable principale
`uat`	Tests utilisateurs	Créer une fois le MVP stable
`feature/xxx`	Nouvelles fonctionnalités	Via Claude Code ou Cursor, merger via PR

Quand épingler une version ?

📌 Épinglez une version quand :

- [ ] Après chaque fonctionnalité qui fonctionne correctement
- [ ] Avant de commencer une modification risquée ou complexe
- [ ] Après un refactoring important
- [ ] Avant de connecter Supabase ou une API externe
- [ ] Avant de changer le schéma de la base de données

⚠️ **Attention aux rollbacks Supabase**

Un rollback de version Lovable n'annule pas les changements de schéma Supabase. Testez soigneusement chaque fonctionnalité liée aux données avant publication.

Supabase : base de données et authentification

Supabase est le backend recommandé de Lovable — PostgreSQL, authentification, stockage et Edge Functions dans un seul service.

> 🎯 **Règle d'or Supabase**
>
> Connectez la base de données **une fois le frontend suffisamment stable**. Ne touchez jamais au schéma trop tôt — les migrations sont coûteuses à corriger.

Connexion Supabase à Lovable

1. Créer un projet sur **supabase.com**
2. Dans Lovable, aller dans **Settings → Integrations → Supabase**
3. Coller l'URL du projet et la clé anonyme
4. Prompter : *« Configure Supabase avec authentification utilisateur »*
5. Lovable configure automatiquement Auth, les tables de base et les politiques RLS initiales

Row Level Security (RLS) — Les bases

Politique	Description	Exemple
SELECT par utilisateur	Un user ne voit que ses données	`auth.uid() = user_id`
INSERT authentifié	Seuls les users connectés créent	`auth.role() = 'authenticated'`
Admin tout-accès	L'admin voit tout	`is_admin(auth.uid())`

Checklist Supabase avant déploiement

🔐 Checklist de sécurité Supabase

- [ ] RLS activé sur toutes les tables contenant des données utilisateur
- [ ] Confirmation email désactivée en dev, réactivée en production
- [ ] Clés API stockées en variables d'environnement (jamais dans le code)
- [ ] .env ajouté au .gitignore pour éviter toute fuite de credentials
- [ ] Branche staging testée avant chaque push en production
- [ ] Security Review Lovable 2.0 lancée avant publication
- [ ] UUIDs utilisés pour toutes les relations entre tables

Intégrations avancées

- **Stripe :** *« Ajoute un bouton de checkout et traite les paiements avec Stripe. »* Lovable crée une Edge Function qui communique avec l'API Stripe.
- **Auth sociale (Google, GitHub) :** Dans Supabase → Auth → Providers, activez OAuth puis demandez à Lovable : *« Ajoute un bouton Sign in with Google. »*
- **Make.com / n8n :** Utilisez des webhooks pour connecter Lovable à des services externes (Slack, CRM, notifications, etc.)

Déploiement : de Lovable à la production

Lovable héberge pour le développement, mais pour la production, externalisez vers Netlify, Vercel ou Cloudflare Pages pour plus de contrôle.

Plateforme	Gratuit	Déploiement auto	Domaine perso	Idéal pour
Lovable.app	Limité	✓	Plans payants	Dev et tests rapides
Cloudflare Pages	✓ Généreux	✓	✓ Inclus	Production recommandée
Vercel	✓	✓	✓	Apps React complexes
Netlify	✓	✓	✓	Sites + forms + fonctions

⚠️ **Publication ≠ Prêt pour la production**

Publier signifie que l'app est accessible. La mise en production nécessite une review complète : contrôle d'accès, sécurité, comportement des données, fiabilité et tests manuels.

Checklist pré-déploiement

🚀 Avant la mise en production

- [ ] Security Review Lovable lancée et validée
- [ ] RLS Supabase activé et testé sur toutes les tables
- [ ] Confirmation email Supabase réactivée
- [ ] Variables d'environnement configurées en production
- [ ] Tests de tous les rôles (Admin, User, Visiteur)
- [ ] Test complet sur mobile (iOS + Android)
- [ ] Flux d'authentification testé end-to-end
- [ ] Domaine personnalisé configuré et HTTPS actif
- [ ] Supabase staging cloné en projet PROD séparé

Débogage : sortir des boucles d'erreur

Les boucles d'erreur sont la principale source de frustration avec Lovable. Voici une méthode systématique pour en sortir.

Le workflow de débogage en 5 étapes

Prompts de débogage essentiels

BUG REPORT EFFICACE

```
Sur /settings, j'attendais [comportement attendu]
mais j'obtiens [comportement observé].

Erreur console :
[Coller l'erreur exacte de la console Chrome]

Fichiers concernés : [Liste les composants visiblement impliqués]

// Ne pas modifier les fichiers non listés.
// Commencer par Plan Mode : identifier sans coder.
```

SORTIR D'UNE BOUCLE D'ERREUR

```
Je suis bloqué sur ce bug depuis [N] tentatives.
Plutôt que de coder, donne-moi 3 approches différentes
pour résoudre ce problème sans toucher au code pour l'instant.

Ce qui a été essayé : [résumé]
Erreur actuelle : [erreur]
```

🔴 **Signes qu'il faut utiliser Remix (repartir proprement)**

Utilisez le bouton
Remix

pour créer une copie propre quand : le même bug revient après 3+ tentatives ; corriger un bug en crée 2 nouveaux ; la dette accumulée coûte plus qu'un redémarrage. Déconnectez Supabase d'abord.

---

11 — ASTUCES

51 astuces pour maîtriser Lovable

Synthèse des meilleures pratiques de la communauté, organisées par thème.

🎯 Prompting & instructions (1–10)

01 — Contexte → Tâche → Contraintes

Structure systématique pour chaque prompt. L'IA privilégie le début et la fin — placez les détails critiques en tête.

02 — Un sujet par prompt

Décomposez vos demandes en étapes testables. Évitez de mélanger 5 modifications sans rapport dans un seul prompt.

03 — Demandez les clarifications avant le code

« Ask me any questions you need before writing any code. » Réduit massivement les allers-retours.

04 — Répétez les instructions importantes

La mémoire du modèle peut être limitée d'un échange à l'autre. Répétez les contraintes critiques régulièrement.

05 — Nommez l'écran et le rôle

Précisez toujours la route (/settings) et le rôle utilisateur concerné pour éviter les effets de bord.

06 — Utilisez du contenu réel dès le départ

Bannissez « lorem ipsum ». Le vrai contenu aide Lovable à mieux structurer l'interface.

07 — Définissez le style visuel tôt

Choisissez un mot de style (calme, premium, expressif, minimal) et répétez-le dans les prompts de design.

08 — Précisez le rôle utilisateur

Si votre app a plusieurs rôles (Admin, User), indiquez toujours lequel est concerné par le prompt.

09 — Bibliothèque de prompts réutilisables

Consignez vos prompts efficaces dans Notion ou un fichier MD. La cohérence des prompts crée de la cohérence dans l'app.

10 — Méta-prompting pour la structure

Demandez à Claude Chat de rédiger votre prompt pour vous si vous ne savez pas par où commencer.

⚙️ Workflow & organisation (11–20)

11 **Plan Mode à 60–70 % du temps**

Déboguer, brainstormer et planifier sans consommer de crédits. Cliquez « Implémenter » uniquement quand satisfait.

12 **GitHub dès le premier MVP**

Activez immédiatement. Chaque modification devient un commit, vous donnant un historique complet et un backup.

13 **Épinglez les versions stables**

Après chaque fonctionnalité qui marche, épinglez. Comparez visuellement T-1 et T0 pour identifier les régressions.

14 **Mobile-first systématique**

Utilisez le prompt responsive mobile-first à chaque nouvel écran. ShadCN + Tailwind natifs uniquement.

15 **Knowledge File à jour**

Le cerveau du projet est envoyé avec chaque prompt. Mettez-le à jour après chaque phase majeure.

16 **Screenshots pour le design**

Capturez un design qui vous plaît et glissez-le dans Lovable pour s'en inspirer. Gain de temps massif.

17 **Figma + Builder.io**

Transformez vos wireframes Figma en projets Lovable pixel-perfect via l'intégration Builder.io.

18 **Remix pour repartir proprement**

Si le projet est trop endommagé, Remix crée une copie propre. Déconnectez Supabase d'abord.

19 **MVP ≤ 5 fonctionnalités core**

Chaque fonctionnalité supplémentaire augmente exponentiellement la complexité. Partez du minimum viable.

20 **Composants atomiques**

« Un formulaire avec un champ email et un bouton arrondi » marche mieux qu'« une section d'inscription ».

🔐 Backend, base de données & sécurité (21–30)

21 **Configurez Supabase en premier**

Verrouillez votre backend avant d'ajouter des features. Prompt : « Configure Supabase avec authentification utilisateur ».

22 **Email de confirmation : off en dev**

Dans Supabase → Auth → Settings, désactivez « Confirm email » pendant le dev. Réactivez avant la production.

23 **RLS sur toutes les tables sensibles**

Lovable configure les bases automatiquement, mais vérifiez manuellement chaque table contenant des données utilisateur.

24 **Branche UAT avant production**

Une fois le MVP stable, créez une branche UAT et clonez votre projet Supabase en PROD pour protéger la production.

25 **Stripe via Edge Functions**

Prompt : « Ajoute un bouton de checkout et traite les paiements avec Stripe. » Lovable crée l'Edge Function.

26 **UUIDs pour les relations**

Vérifiez que vos clés étrangères correspondent bien aux types de vos clés primaires pour éviter les erreurs silencieuses.

27 **Auth sociale en 5 minutes**

Supabase → Auth → Providers → activer OAuth, puis : « Ajoute un bouton Sign in with Google ».

28 **Toggle staging/production**

Lovable 2.0 intègre un toggle natif staging/production. Testez toujours sur staging avant de pousser en prod.

29 **Secrets en variables d'env**

Jamais de clés API dans le code. Settings → Environment Variables, et .env dans .gitignore.

30 **Security Review avant déploiement**

Lovable 2.0 inclut un scan IA de sécurité. Utilisez-le systématiquement et activez la détection de fuites de clés.

🐛 Débogage & erreurs (31–40)

31 **Console Chrome en premier**

F12 → logs console, network requests, erreurs JS. Toujours avant de lancer un nouveau prompt de correction.

32 **Format bug report : attendu vs obtenu**

« Sur /page, j'attendais X mais j'obtiens Y. Erreur console : [erreur]. » La précision réduit les tentatives.

33 **Sortir des boucles : 3 approches**

Si l'IA boucle : 1) 3 approches sans coder, 2) rollback vers version épinglée, 3) reconstruire from scratch.

34 **Refactorisez régulièrement**

« Refactorise ce fichier sans changer l'UI ni la fonctionnalité. Documente les changements et implémente graduellement. »

35 **Plan Mode pour diagnostiquer**

« Identifie le problème mais n'écris pas de code. » Vous restez maître de la solution appliquée.

36 **Maintenez un filesExplainer.md**

Document à jour décrivant vos composants. L'IA évite de casser ce qu'elle ne connaît pas.

37 **Testez tous les rôles après chaque change**

Un composant partagé modifié peut casser plusieurs vues. Vérifiez Admin, User, Visiteur.

38 **Composants spécifiques par rôle**

DashboardAdmin, DashboardUser plutôt qu'un composant partagé. Prévient les effets de bord.

39 **« Try to Fix » avec discernement**

Si « Try to Fix » ne résout pas après 2–3 tentatives, arrêtez. Reformulez avec le contexte exact.

40 **Screenshots pour les bugs UI**

Uploadez une capture d'écran du bug. « Why is this UI behaving this way? » est très efficace.

🎨 UI, design & Visual Edits (41–45)

41 **Visual Edits pour le pixel-perfect**

Gratuit et instantané : cliquez n'importe quel élément pour le modifier directement comme dans Figma, sans prompt.

42 **Image de référence UI**

Joignez une image de référence à votre prompt. L'IA interprète les inputs visuels plus vite que les descriptions.

43 **ShadCN + Tailwind comme base**

Ces librairies sont parfaitement intégrées à Lovable. Spécifiez-les explicitement pour un code maintenable.

44 **Composants dédiés par rôle**

Pour les multi-rôles, créez des composants dédiés. Évite la complexité et les bugs de composants partagés.

45 **Dev Mode pour l'édition code-first**

Disponible sur les plans payants : éditez directement le TypeScript/React pour les ajustements fins.

🚀 Performance, déploiement & communauté (46–51)

46 **Supabase scale sans réécriture**

Lovable + Supabase supporte des milliers d'utilisateurs. Planifiez la structure de données dès le début.

47 **MVP d'abord, features ensuite**

3 à 5 fonctionnalités core maximum pour la V1. Chaque ajout augmente exponentiellement la complexité.

48 **Netlify/Vercel pour la production**

Connectez votre repo GitHub pour des déploiements automatiques. Lovable héberge le dev, externalisez la prod.

49 **Discord Lovable**

Mine d'or : prompts partagés par des Champions, solutions aux problèmes communs, accès aux bêtas.

50 **Itérez comme une conversation**

Le meilleur résultat vient après 2–3 itérations. Traitez Lovable comme un partenaire créatif, pas un oracle.

51 **⭐ Lovable + Claude Code via GitHub**

Le combo qui divise les coûts par 10. Lovable pour la génération et les previews, Claude pour la logique et le debug. La communauté rapporte –90 % de crédits Lovable.

12 — WORKFLOW AVANCÉ

Lovable + Claude Code : le combo gagnant

Le workflow le plus puissant de la communauté dev IA — Lovable pour la vitesse, Claude pour la profondeur.

🏆 La logique du combo

Lovable pour la génération initiale et les previews — irremplaçable. Claude Code pour la logique, le debug et le refactoring — sans système de crédits. En les combinant via GitHub, vous obtenez la rapidité du no-code et la puissance du code assisté par IA.

Le workflow concret, étape par étape

1. **Dans Lovable :** Cliquez sur le bouton GitHub pour synchroniser votre projet. Dès lors, chaque modification devient un commit automatique.
2. **Clonez le repo :** Localement ou directement dans Claude Code / Cursor via l'interface GitHub.
3. **Dans Claude Code :** Connectez GitHub, sélectionnez le repo Lovable, créez une nouvelle branche. Demandez vos modifications — debug, refactoring, logique backend — sans consommer un seul crédit Lovable.
4. **Pull Request :** Mergez la branche via une PR GitHub.
5. **Lovable détecte les commits :** Reconstruit automatiquement le preview — vous voyez le résultat dans Lovable sans avoir rien généré de son côté.

Répartition des rôles

Outil	Utilisation optimale	À éviter
Lovable	Génération initiale, prototypage UI rapide, previews temps réel, déploiement	Refactoring massif, logique métier complexe
Claude Chat / Plan Mode	Brainstorming, rédaction de PRD, debug complexe, conception du schéma	Génération de code direct sans validation
Claude Code	Refactoring, logique backend, corrections précises, optimisations	Prototypage UI (pas de preview visuel)
Cursor + Claude	Édition code-first, gestion API, agents automatisés	Premiers prototypes visuels

Templates de prompts prêts à l'emploi

Copiez-collez ces templates directement dans Lovable en adaptant les sections entre crochets.

🚀 Lancer un nouveau projet

PROMPT DE DÉMARRAGE DE PROJET — Niveau 1

```
# Contexte
Je construis [type d'application] pour [public cible].
Objectif : [ce que l'app doit faire en 1 phrase].

## Stack technique
- Frontend : React 18 + TypeScript
- Styling : Tailwind CSS + ShadCN/UI
- Backend : Supabase (PostgreSQL + Auth)
- Mobile-first, breakpoints ShadCN natifs

## Fonctionnalités core (MVP uniquement)
1. [Fonctionnalité 1]
2. [Fonctionnalité 2]
3. [Fonctionnalité 3]

## Exclusions explicites
- Pas de [exclusion 1]
- Pas de [exclusion 2]
- Aucune donnée fictive ou mockée

## Style visuel
[calme / premium / expressif / minimal]
Couleur primaire : [#hex]

## Première étape
Commence par la page d'accueil contenant :
[Description précise du premier écran]
```

🗄️ Connexion Supabase

CONFIGURATION SUPABASE INITIALE

```
Configure Supabase avec :
1. Authentification utilisateur (email + mot de passe)
2. Tableau de bord utilisateur sécurisé
3. Row Level Security activé sur toutes les tables
4. Protection des routes : redirige vers /login si non connecté
5. Ne pas créer de données mockées ou de faux utilisateurs

Rôles :
- Admin : accès complet + gestion des utilisateurs
- User : accès à ses données uniquement

Ne modifier que les fichiers nécessaires à l'auth.
Pas de refactorisation globale.
```

🐛 Correction de bug ciblée

BUG FIX PRÉCIS

```
Problème : Sur la page [/route], quand [action],
[comportement observé] au lieu de [comportement attendu].

Erreur console :
[Coller le message d'erreur exact]

Instructions :
1. Commence par analyser la cause racine en Plan Mode
2. Propose ta solution AVANT d'écrire du code
3. Ne modifier QUE le composant [NomComposant]
4. Ne pas toucher aux fichiers [liste]
5. Tester que les autres fonctionnalités restent intactes
```

✅ Non-régression

PROMPT DE VÉRIFICATION FINALE

```
Effectue une vérification complète de non-régression :

1. Vérifie que le périmètre n'a pas été modifié
2. Vérifie qu'aucune colonne de données n'a été renommée
3. Vérifie que toutes les règles métier listées sont respectées
4. Vérifie les scénarios utilisateur : [liste]
5. Vérifie les états : chargement / erreur / vide / succès
6. Vérifie qu'aucune fonctionnalité parasite n'a été ajoutée
7. Vérifie que la logique métier n'est pas dans les composants UI

Rapport de conformité :
- ✅ Conforme
- ❌ Non conforme + correction nécessaire
- ⚠️ Point d'attention
```

🔄 Refactoring sûr

REFACTORING SANS RÉGRESSION

```
Refactorise [composant/fichier] en respectant ces règles :
- Ne pas changer l'UI visible
- Ne pas changer la fonctionnalité
- Ne pas changer les API calls ni le state
- Documenter TOUS les changements effectués
- Implémenter graduellement, section par section
- Arrêter et alerter si un risque de régression est détecté

Objectif : [améliorer la lisibilité / réduire la duplication /
optimiser les performances / etc.]
```

14 — FAQ

Questions fréquentes

Les réponses aux questions les plus posées par la communauté Lovable.

Faut-il savoir coder pour utiliser Lovable ?

Non, mais « accessible » ne veut pas dire « sans méthode ». La maîtrise de Lovable vient de la rigueur dans la préparation du projet et dans la structure des prompts, pas de la compétence technique. Des notions de base en web sont un plus pour déboguer, mais pas nécessaires pour démarrer.

Le plan gratuit est-il suffisant ?

Le plan gratuit (5 crédits) peut suffire pour tester le produit et apprendre le workflow. En revanche, pour un développement produit continu, il sera insuffisant rapidement. Le plan Starter (~20 $/mois) est le minimum recommandé pour un premier projet sérieux.

Que faire si Lovable ne comprend pas ma demande ?

Décomposez votre demande. Si Lovable ne comprend pas, c'est presque toujours que la demande est trop vague ou trop large. Activez le Plan Mode, demandez-lui de poser des questions de clarification, puis reformulez avec un seul objectif précis par prompt.

Peut-on modifier le code généré manuellement ?

Oui, de deux façons : via le Dev Mode (disponible sur les plans payants), qui permet d'éditer directement le TypeScript/React, ou via GitHub en clonant le repo et en utilisant votre éditeur préféré. Les modifications externes sont détectées et reconstruites automatiquement dans Lovable.

Lovable gère-t-il la sécurité des données ?

Lovable fournit des outils (Security Review, scan de sécurité, configuration RLS automatique), mais cela ne dispense pas de tests manuels. Pour une application de production, vérifiez systématiquement les politiques RLS, les variables d'environnement et testez tous les rôles utilisateur.

Quand passer de Lovable à GitHub ou un IDE classique ?

Passez à une approche plus poussée quand la précision, les tests, la collaboration ou une personnalisation avancée priment sur la vitesse. En pratique, de nombreuses équipes procèdent ainsi une fois que le produit a validé la demande ou est entré en phase d'itération régulière.

Lovable est-il adapté aux applications de production ?

Oui, mais avec le même sérieux que toute autre application : review backend, validation sécurité, tests des autorisations, gestion des dépendances. La publication Lovable signifie seulement que l'app est accessible — la mise en production nécessite un processus complet de validation.

Comment éviter les « hallucinations » de Lovable ?

La méthode en 6 phases est la réponse principale : verrouiller le périmètre, décrire contractuellement les données, écrire explicitement chaque règle métier. La règle absolue : « Si une règle n'est pas écrite explicitement, elle n'existe pas. »

Lovable peut-il générer à partir d'une image ou d'un screenshot ?

Oui. Vous pouvez uploader des captures d'écran de sites existants, des maquettes Figma ou des croquis Excalidraw directement dans Lovable pour guider le design. L'IA interprète les inputs visuels et génère un code inspiré de la référence fournie.

Le plan payant vaut-il l'investissement ?

Si vous développez et validez activement un produit ou un flux de travail réel, généralement oui. La question n'est pas tant le coût mensuel que la vitesse à laquelle Lovable vous permet d'atteindre un objectif produit validé par rapport aux autres méthodes de travail.

15 — WORKFLOW BOUT-EN-BOUT

Cas concret : Application SaaS de réservation

Construction complète d'un outil de gestion de réservations pour une salle de sport — de l'idée au déploiement en production, prompt par prompt.

🎯 Le projet

FitBook — Application de réservation de créneaux pour une salle de sport. Les membres réservent des cours en ligne. Les coaches voient leur planning. L'admin gère tout. Paiement via Stripe.

Phase 0 — Contexte immuable

PROMPT 0 — CONTEXTE STRATÉGIQUE

Aucun code généré

```
Projet : FitBook — Application de réservation de créneaux pour salle de sport.

Public cible :
- Membres (clients) : réservent des cours, voient leur historique
- Coaches : voient leur planning hebdomadaire
- Admin : gère cours, créneaux, membres et paiements

Niveau : MVP — pas de gamification, pas d'app mobile native

Exclusions explicites :
- Pas de système de fidélité dans cette version
- Pas de chat en temps réel
- Pas de notifications push (emails uniquement)
- Pas de multi-salles dans le MVP

Contrainte forte :
Les données de réservation sont contractuelles.
Aucun renommage de colonne sans validation explicite.
```

Phase 1 — Architecture & navigation

PROMPT 1 — STRUCTURE DE L'APPLICATION

Étape 1 — Architecture de FitBook

Objectif : Définir la structure globale sans écrire de code métier.

Pages à créer :
/ → Landing page publique (présentation + login/signup)
/login → Connexion
/signup → Inscription membre
/dashboard → Tableau de bord membre (réservations à venir)
/cours → Liste des cours disponibles avec filtres
/reservation/:id → Détail + réservation d'un cours
/mon-compte → Profil, historique, abonnement
/coach/planning → Vue planning du coach (accès restreint)
/admin → Dashboard admin
/admin/cours → Gestion des cours et créneaux
/admin/membres → Liste et gestion des membres

Navigation :
- Header avec logo, nav principale, bouton profil
- Sidebar admin collapsible
- Mobile : bottom navigation pour membres

Interdictions :
- Pas de logique métier dans cette étape
- Pas de données réelles, uniquement la structure
- Ne pas créer les composants de formulaire

Validation :
Génère uniquement la structure de fichiers et le routeur React.
Confirme avant d'implémenter.

Phase 2 — Schéma de données Supabase

PROMPT 2 — DONNÉES CONTRACTUELLES

Étape 2 — Schéma Supabase pour FitBook

Tables et colonnes contractuelles (ne jamais renommer) :

```
TABLE profiles
  id: uuid (FK → auth.users, non-null)
  role: enum('member','coach','admin') default 'member'
  full_name: text (non-null)
  phone: text (nullable)
  avatar_url: text (nullable)
  stripe_customer_id: text (nullable)
  created_at: timestamptz default now()

TABLE courses
  id: uuid (PK, default gen_random_uuid())
  title: text (non-null)
  description: text (nullable)
  coach_id: uuid (FK → profiles.id, non-null)
  capacity: integer (non-null, default 10)
  duration_minutes: integer (non-null)
  category: enum('yoga','crossfit','cardio','pilates','force')
  created_at: timestamptz default now()

TABLE slots
  id: uuid (PK)
  course_id: uuid (FK → courses.id, non-null)
  starts_at: timestamptz (non-null)
  ends_at: timestamptz (non-null)
  available_spots: integer (non-null)
  is_cancelled: boolean default false

TABLE bookings
  id: uuid (PK)
  slot_id: uuid (FK → slots.id, non-null)
  member_id: uuid (FK → profiles.id, non-null)
  status: enum('confirmed','cancelled','waitlist') default 'confirmed'
  stripe_payment_intent_id: text (nullable)
  booked_at: timestamptz default now()
```

Règles RLS obligatoires :

```
- profiles : SELECT par owner (auth.uid() = id) + admin all
- courses : SELECT public, INSERT/UPDATE coach+admin
- slots : SELECT public, INSERT/UPDATE admin
- bookings : SELECT par member owner + coach du cours + admin
             INSERT par member authentifié uniquement
```

Générer :

Le SQL de création des tables + politiques RLS.
Ne pas encore connecter à l'UI.

Phase 3 — Logique métier

PROMPT 3 — RÈGLES MÉTIER EXPLICITES

```
Étape 3 — Règles métier FitBook (à implémenter strictement)

R1 : Un membre ne peut réserver un même créneau qu'une seule fois.
R2 : available_spots décrémente de 1 à chaque réservation confirmée.
R3 : Si available_spots = 0, proposer la liste d'attente (status='waitlist').
R4 : Une annulation par un membre libère 1 place ET notifie le 1er
     membre en liste d'attente (email via Supabase Edge Function).
R5 : Un cours annulé (is_cancelled=true) annule automatiquement
     toutes les réservations associées et rembourse via Stripe.
R6 : Un coach ne peut voir que ses propres créneaux.
R7 : L'admin peut tout voir et tout modifier.
R8 : Aucune réservation possible moins de 2h avant le début du créneau.

Implémenter via :
- Supabase Edge Functions pour R4 et R5 (logique serveur)
- Triggers Postgres pour R1 et R2 (contrainte base de données)
- Validation côté client pour R8 (UX uniquement, pas de sécurité)

Interdictions :
- Ne pas mettre R1-R5 uniquement dans le frontend
- Ne pas oublier de gérer les cas d'erreur côté UI
```

Phase 4 — Interface utilisateur

PROMPT 4 — UI MEMBRE : LISTE DES COURS

Étape 4a — Page /cours (vue membre)

Style : Moderne, énergique, couleurs primaires #6C47FF et #FF5C4D

Composants à créer :

1. CourseCard : image, titre, coach, catégorie, durée, places dispo
   - Badge "Complet" si available_spots = 0
   - Badge "Liste d'attente" au survol si complet
   - Badge "Bientôt" si démarre dans moins de 24h

2. CourseFilters : filtres par catégorie (chips), par date (datepicker), par coach (dropdown) — tous cumulables

3. CourseGrid : grille 3 colonnes desktop, 1 colonne mobile

4. EmptyState : message si aucun cours correspond aux filtres

États obligatoires :

- Chargement : skeleton loader sur les cards
- Erreur réseau : message + bouton "Réessayer"
- Vide : illustration + message "Aucun cours disponible"

Interdictions :

- Ne pas modifier le schéma de données
- Ne pas ajouter de fonctionnalités hors scope
- Mobile-first, breakpoints Tailwind natifs

Phase 5 — Vérification & non-régression

PROMPT 5 — AUDIT DE CONFORMITÉ FITBOOK

```
Effectue un audit complet de FitBook avant déploiement :

1. DONNÉES
   ✓ Toutes les colonnes contractuelles sont présentes et non renommées ?
   ✓ Les enums utilisent bien les valeurs définies ?
   ✓ Les FK pointent vers les bonnes tables ?

2. SÉCURITÉ
   ✓ RLS activé sur les 4 tables ?
   ✓ Un membre peut-il accéder aux réservations d'un autre membre ?
   ✓ Un coach peut-il modifier des cours qu'il ne possède pas ?
   ✓ Les clés Stripe sont bien en variables d'environnement ?

3. LOGIQUE MÉTIER
   ✓ R1 à R8 sont-ils tous implémentés côté serveur ?
   ✓ Le décrément de available_spots est-il atomique (trigger) ?
   ✓ L'annulation déclenche bien le remboursement Stripe ?

4. UI
   ✓ Tous les états (loading/error/empty) sont présents ?
   ✓ L'app est responsive sur mobile 375px ?
   ✓ Les badges de statut s'affichent correctement ?

Produis un rapport ✅/❌/⚠️ pour chaque point.
```

Résumé du timeline réaliste

Phase	Durée estimée	Crédits estimés
Phase 0 — Contexte	30 min (Plan Mode)	0
Phase 1 — Architecture	1h	3–5
Phase 2 — Données	1h	4–6
Phase 3 — Logique métier	2h	6–10
Phase 4 — UI complète	3–4h	10–15
Phase 5 — Audit + fix	1–2h	5–8
Total MVP	**8–12h**	**28–44 crédits**

16 — WORKFLOW BOUT-EN-BOUT

Cas concret : Site internet professionnel

Création d'un site vitrine pour un cabinet d'architectes — landing page, portfolio, blog, formulaire de contact, SEO optimisé.

🎯 Le projet

ArchStudio — Site vitrine pour un cabinet d'architectes indépendants. Objectif : attirer des clients particuliers et entreprises via Google. Blog pour le SEO. Formulaire de contact avec qualification du lead.

Prompt de démarrage site vitrine

PROMPT COMPLET — SITE VITRINE AVEC SEO

```
Projet : Site vitrine pour cabinet d'architectes ArchStudio.

Stack :
React 18 + TypeScript + Tailwind + ShadCN
Supabase pour le blog (CMS headless) et les contacts
Déploiement : Cloudflare Pages

Pages à créer :
/ → Hero + services + projets phares + témoignages + CTA
/projets → Portfolio avec filtres (résidentiel/commercial/réhabilitation)
/projets/:slug → Fiche projet avec galerie photos, description, méta SEO
/blog → Liste articles avec pagination
/blog/:slug → Article complet avec méta SEO dynamique
/contact → Formulaire qualifié (type de projet, budget, localisation)
/a-propos → Équipe, valeurs, certifications

SEO — Obligatoire sur chaque page :
- title dynamique : "[Titre page] | ArchStudio Architectes Paris"
- meta description unique de 150-160 caractères
- og:title, og:description, og:image
- canonical URL
- JSON-LD Schema.org (Organization sur /, Article sur /blog/:slug,
  LocalBusiness sur /contact)

Style :
Minimaliste premium. Blanc cassé + noir + une touche cuivre (#B87333).
Typographie : Georgia pour les titres, Arial pour le corps.
Animations subtiles au scroll (pas de parallax agressif).

Performance (Core Web Vitals) :
- Images en format WebP avec lazy loading
- Fonts : system fonts uniquement (pas de Google Fonts en prod)
- Code splitting par route (lazy imports React)

Première étape uniquement :
La page d'accueil complète avec Hero, section Services (3 cartes),
et section Projets phares (grille 3x2).
Contenu réel fourni ci-dessous — pas de lorem ipsum.
```

Prompt — Formulaire de contact qualifié

PROMPT — FORMULAIRE AVEC QUALIFICATION LEAD

```
Crée le formulaire de contact /contact avec ces champs :

Étape 1 (Type de projet) :
- Type : checkbox multiple (Résidentiel neuf / Réhabilitation /
  Commercial / Autre)

Étape 2 (Budget et localisation) :
- Budget estimé : select (< 100k€ / 100-300k€ / 300k-1M€ / > 1M€)
- Localisation : text (ville + département)
- Délai souhaité : select (Urgent < 3 mois / 3-12 mois / > 1 an)

Étape 3 (Contact) :
- Prénom + Nom : text (non-null)
- Email : email (non-null, validation format)
- Téléphone : tel (nullable)
- Message : textarea (min 50 caractères)

Comportement :
- Formulaire multi-étapes avec barre de progression
- Validation en temps réel à chaque champ
- Soumission → INSERT dans table Supabase 'leads'
- Email de confirmation automatique via Supabase Edge Function
- Email de notification admin (adresse en variable d'environnement)
- Page de confirmation avec CTA vers portfolio

Table Supabase 'leads' (colonnes contractuelles) :
  id: uuid, project_types: text[], budget_range: text,
  location: text, timeline: text, full_name: text (non-null),
  email: text (non-null), phone: text, message: text,
  created_at: timestamptz, status: enum('new','contacted','qualified','lost')

RLS : INSERT public, SELECT admin uniquement.
```

Prompt — Blog CMS avec Supabase

PROMPT — BLOG HEADLESS CMS

```
Configure le blog avec Supabase comme CMS headless.

Table 'posts' (colonnes contractuelles) :
  id: uuid, slug: text (unique, non-null),
  title: text (non-null), excerpt: text (150 chars max),
  content: text (Markdown), cover_image_url: text,
  author_id: uuid (FK → profiles.id),
  published_at: timestamptz (null = brouillon),
  meta_title: text, meta_description: text (160 chars max),
  tags: text[], reading_time_minutes: integer

Table 'tags' : id uuid, name text, slug text unique

Pages à implémenter :
1. /blog → Grille d'articles paginée (10/page), filtre par tag
   - Meta : "Blog Architecture | ArchStudio"
2. /blog/:slug → Article complet
   - Meta dynamique depuis meta_title et meta_description
   - Schema.org Article en JSON-LD
   - Table des matières générée depuis les H2/H3 du contenu
   - Articles recommandés (même tag, 3 maximum)

Interdictions :
- Pas d'éditeur WYSIWYG dans le MVP (le contenu est géré
  directement dans Supabase)
- Pas de commentaires dans cette version
- Pas de newsletter dans cette version
```

Cas concret : Outil métier interne — CRM PME

Construction d'un CRM léger pour une PME de 15 personnes — gestion des clients, suivi des devis, pipeline commercial, tableau de bord KPI.

🎯 Le projet

PipeLine — CRM interne pour une PME de services B2B. Remplacement de fichiers Excel. Accès web depuis n'importe où. Pas de code, pas d'IT interne. Budget : plan Starter Lovable.

Prompt de démarrage CRM

PROMPT 0+1 COMBINÉS — CRM PME

```
Projet : PipeLine — CRM interne pour PME de services B2B.

Utilisateurs :
- Commercial : gère ses clients et devis, voit son pipeline
- Manager : voit tout, génère les rapports
- (Pas d'admin séparé dans le MVP)

Modules MVP :
1. Clients : fiche client, historique interactions, statut
2. Contacts : personnes liées à un client (plusieurs contacts/client)
3. Opportunités : montant, probabilité, étape pipeline, closing date
4. Activités : appels, emails, RDV liés à une opportunité
5. Dashboard : KPIs commercial + manager

Exclusions MVP :
- Pas de génération de PDF de devis
- Pas d'envoi d'emails depuis l'app
- Pas d'import CSV dans le MVP
- Pas de mobile app native

Pipeline commercial (étapes fixes) :
Prospect → Qualifié → Proposition → Négociation → Gagné/Perdu

Commencer par :
La structure de fichiers + le schéma Supabase uniquement.
Présente le schéma avant d'implémenter.
```

Prompt — Dashboard KPI manager

PROMPT — DASHBOARD AVEC KPIS

```
Crée le dashboard manager /dashboard/manager avec ces KPIs :

Ligne 1 — Métriques globales (4 cartes) :
- CA pipeline total (somme opportunities.amount où status != 'lost')
- Opportunités actives (count, hors 'won'/'lost')
- Taux de conversion (won / total fermés × 100, en %)
- Deal moyen (CA gagné / count won, période glissante 90j)

Ligne 2 — Pipeline visuel :
- Kanban readonly (colonnes = étapes pipeline)
- Chaque card : client, montant, commercial, probabilité
- Couleur de la card selon la probabilité :
  vert > 70%, orange 40-70%, rouge < 40%

Ligne 3 — Tableau des commerciaux :
Colonnes : Nom | Opps actives | CA pipeline | CA gagné (90j) | Conv. %
Tri par CA gagné décroissant

Règles :
- Toutes les données viennent de Supabase (pas de mock)
- Filtres : période (7j/30j/90j/personnalisé) + commercial
- Actualisation automatique toutes les 5 minutes
- RLS : visible par manager uniquement

États obligatoires :
- Skeleton loader pendant le chargement
- Message si aucune donnée sur la période sélectionnée
```

Prompt — Import CSV clients existants

PROMPT — IMPORT DE DONNÉES EXISTANTES

```
Ajoute une page d'import CSV /admin/import pour migrer
les clients existants depuis Excel.

Format CSV attendu (colonnes contractuelles) :
  company_name, siret, industry, website, city, postal_code,
  country (default 'FR'), annual_revenue, employee_count,
  main_contact_name, main_contact_email, main_contact_phone,
  commercial_name, status (prospect/client/inactive), notes

Comportement :
1. Upload du fichier CSV
2. Prévisualisation des 5 premières lignes avec validation
3. Détection des erreurs : email invalide, SIRET non numérique,
   doublon sur company_name + siret
4. Rapport avant import : X lignes valides, Y erreurs détectées
5. Confirmation explicite avant INSERT en base
6. Import en transaction (tout ou rien)
7. Log des imports dans table 'import_logs'

Interdictions :
- Pas d'import si des erreurs bloquantes sont détectées
- Pas de modification silencieuse des données du CSV
- Ne jamais écraser un client existant sans confirmation
```

Sécurité avancée & architecture à vérifier

La sécurité n'est pas une fonctionnalité — c'est une couche transversale. Voici ce qu'il faut vérifier absolument avant toute mise en production.

Les 4 menaces principales à connaître

🔴 Injection SQL & RLS bypass

- Table sans politique RLS = toutes les données lisibles
- Paramètres non sanitisés dans les requêtes
- Service role key exposée côté client
- Edge Functions sans vérification du token JWT

🟠 Exposition de credentials

- Clés API en dur dans le code source
- Fichier .env commité dans GitHub
- Clé Stripe publishable vs secret key confondue
- Variables Supabase service_role côté frontend

🟢 Authentification défaillante

- Routes protégées accessibles sans token
- Token JWT non vérifié dans les Edge Functions
- Session persistante sans expiration
- Pas de rate limiting sur /login

🟣 Fuite de données côté client

- Données sensibles dans l'URL (query params)
- Console.log de tokens en production
- Réponses API trop verbeuses (données non demandées)
- LocalStorage pour stocker des tokens sensibles

Checklist RLS complète — Supabase

🔐 Audit RLS — À vérifier table par table

- [ ] RLS activé sur **chaque table** — vérifier dans Supabase → Table Editor → RLS
- [ ] Politique SELECT : un user ne peut lire que ses propres données (`auth.uid() = user_id`)
- [ ] Politique INSERT : seuls les utilisateurs authentifiés peuvent créer (`auth.role() = 'authenticated'`)
- [ ] Politique UPDATE/DELETE : uniquement sur ses propres enregistrements
- [ ] Rôle admin vérifié via fonction `is_admin()` ou colonne `profiles.role`
- [ ] Tester en mode "incognito" avec un compte user standard : aucun accès admin possible
- [ ] Tester avec 2 comptes différents : user A ne peut pas voir les données de user B
- [ ] Edge Functions : vérifier le JWT avec `supabase.auth.getUser(token)` avant tout traitement

Prompt — Audit de sécurité Lovable

PROMPT D'AUDIT SÉCURITÉ COMPLET

```
Effectue un audit de sécurité complet de l'application :

1. SUPABASE RLS
   - Liste toutes les tables sans politique RLS active
   - Identifie les politiques trop permissives (SELECT without auth)
   - Vérifie que service_role n'est pas utilisé côté client

2. CREDENTIALS
   - Recherche toute clé API, token ou secret en dur dans le code
   - Vérifie que .env est dans .gitignore
   - Liste toutes les variables d'environnement nécessaires

3. ROUTES & AUTORISATION
   - Liste toutes les routes protégées
   - Vérifie que chaque route vérifie l'authentification
   - Vérifie que les routes admin vérifient le rôle admin

4. EDGE FUNCTIONS
   - Vérifient-elles le JWT entrant ?
   - Les erreurs retournent-elles des messages génériques
     (pas de stack trace en production) ?

5. DONNÉES SENSIBLES
   - Y a-t-il des données sensibles dans les URL ?
   - Des console.log de données sensibles en production ?

Rapport attendu : liste de vulnérabilités par niveau
CRITIQUE / ÉLEVÉ / MOYEN / FAIBLE + correction recommandée.
```

Architecture recommandée pour une app de production

Couche	Responsabilité	Règle de sécurité
Frontend React	Affichage, UX, validation côté client	Jamais de logique de sécurité uniquement côté client
Supabase Auth	JWT, sessions, OAuth	Vérifier le token dans chaque Edge Function
Supabase RLS	Contrôle d'accès aux données	Toujours actif, jamais désactivé en production
Edge Functions	Logique sensible, Stripe, emails	Variables d'env pour tous les secrets
PostgreSQL	Persistance + contraintes	Contraintes DB comme filet de sécurité ultime

🚨 **Ne jamais utiliser la service_role key côté client**

La clé `service_role` Supabase bypass toutes les politiques RLS. Elle ne doit exister qu'en variable d'environnement serveur (Edge Functions). La clé `anon` est la seule à utiliser côté frontend.

Checklist architecture complète

🏗️ Architecture — Checklist avant production

- [ ] **Séparation des responsabilités** — La logique métier est dans les Edge Functions, pas dans les composants React
- [ ] **Single source of truth** — Les données viennent toujours de Supabase, jamais d'un état local non synchronisé
- [ ] **Error boundaries** — Chaque section critique a un composant ErrorBoundary React
- [ ] **Loading states** — Chaque appel async a un état de chargement visible
- [ ] **Optimistic updates** — Les mutations mettent à jour l'UI immédiatement puis confirment
- [ ] **Pagination** — Aucune requête ne ramène l'intégralité d'une table (toujours LIMIT)
- [ ] **Index DB** — Les colonnes utilisées dans les WHERE et JOIN sont indexées
- [ ] **Migrations versionnées** — Tous les changements de schéma sont dans des fichiers SQL versionnés
- [ ] **Variables d'env** — 0 secret dans le code, tous en variables d'environnement
- [ ] **CORS configuré** — Les Edge Functions n'acceptent que les origines autorisées

Gestion des mots de passe & authentification

Supabase Auth gère la cryptographie — mais vous devez configurer correctement les politiques, les flux et les mécanismes de récupération.

Configuration Supabase Auth recommandée

Paramètre	Valeur recommandée	Où configurer
Confirmation email	**OFF en dev, ON en prod**	Auth → Settings → Email
Expiration JWT	3600s (1h) en prod	Auth → Settings → JWT
Refresh token rotation	**Activé**	Auth → Settings → Security
Minimum password length	8 caractères minimum	Auth → Settings → Password
Rate limiting login	**Activé par défaut**	Supabase géré automatiquement
MFA (TOTP)	Optionnel mais recommandé pour admin	Auth → Multi-Factor Auth

Prompt — Flux d'authentification complet

PROMPT — AUTH COMPLÈTE AVEC RESET PASSWORD

```
Implémente le flux d'authentification complet avec Supabase :

1. INSCRIPTION (/signup)
   Champs : email, mot de passe, confirmation, prénom, nom
   Validation :
   - Email : format valide
   - Mot de passe : min 8 caractères, 1 majuscule, 1 chiffre, 1 symbole
   - Confirmation : identique au mot de passe
   Après soumission : email de confirmation Supabase
   Message : "Vérifiez votre boîte email pour confirmer votre compte"

2. CONNEXION (/login)
   Champs : email, mot de passe
   Option : "Se souvenir de moi" (session persistante 30j)
   Lien "Mot de passe oublié" → /reset-password
   Erreurs : message générique "Email ou mot de passe incorrect"
   (ne jamais révéler si l'email existe)

3. RESET MOT DE PASSE (/reset-password)
   Étape 1 : email uniquement → envoie magic link Supabase
   Étape 2 : nouveau mot de passe + confirmation (depuis le lien email)
   Validation : mêmes règles que l'inscription
   Succès : redirection vers /login avec message de confirmation

4. PROTECTION DES ROUTES
   HOC ProtectedRoute : vérifie session Supabase
   AdminRoute : vérifie profiles.role = 'admin'
   Redirection vers /login si non authentifié

5. DÉCONNEXION
   Bouton dans le menu profil
   supabase.auth.signOut() + redirection vers /
   Nettoyage du state local

Interdictions :
- Ne jamais stocker le mot de passe en clair ou hashé localement
- Ne jamais logger les tokens JWT en console
- Ne jamais afficher si l'email est déjà enregistré
```

Prompt — Authentification sociale (OAuth)

PROMPT — OAUTH GOOGLE + GITHUB

```
Ajoute l'authentification sociale sur /login et /signup :

Providers à activer :
- Google OAuth (pré-requis : app configurée dans Google Console)
- GitHub OAuth (pré-requis : app configurée dans GitHub Settings)

UI :
- Boutons séparés : "Continuer avec Google" + "Continuer avec GitHub"
- Séparateur "ou" entre OAuth et formulaire email
- Icônes officielles des providers (SVG inline)

Comportement :
- Callback URL : /auth/callback
- À la première connexion OAuth : création automatique du profil
  dans la table 'profiles' avec role='member'
- Si l'email existe déjà (compte email classique) :
  proposer de lier les comptes (message d'information)

Sécurité :
- Le callback /auth/callback vérifie le code et crée la session
- Redirection vers /dashboard après succès
- Redirection vers /login?error=oauth_failed en cas d'échec

Configurer dans Supabase :
Authentication → Providers → Google/GitHub → activer
Redirect URL : https://[votre-projet].supabase.co/auth/v1/callback
```

Politique de mots de passe recommandée

🔑 Politique de mots de passe — Standards 2026

→ Longueur minimum : 8 caractères (12 recommandé pour les admins)

→ Complexité : au moins 1 majuscule, 1 chiffre, 1 caractère spécial

→ Interdire les mots de passe courants (liste NIST) — Supabase le gère

→ Jamais stocker le mot de passe, uniquement le hash bcrypt (Supabase natif)

→ Expiration de session : 1h (JWT) avec refresh token de 30j

→ Rotation des refresh tokens à chaque utilisation

→ MFA TOTP obligatoire pour les comptes admin en production

→ Verrouillage temporaire après 5 tentatives échouées (géré par Supabase)

Activer le MFA pour les admins

PROMPT — MFA POUR LES COMPTES ADMIN

```
Implémente le MFA (authentification à deux facteurs) TOTP
pour les utilisateurs avec role='admin'.

Flux d'enrôlement :
1. À la première connexion admin : popup obligatoire d'activation MFA
2. Afficher le QR code Supabase MFA (Google Authenticator / Authy)
3. Demander la saisie du code TOTP pour confirmer l'enrôlement
4. Afficher et forcer la sauvegarde des codes de récupération

Flux de connexion admin :
1. Email + mot de passe (étape 1)
2. Code TOTP 6 chiffres (étape 2)
3. Option "Faire confiance à cet appareil 30j"

Flux de récupération :
- Bouton "Utiliser un code de récupération" sur l'étape TOTP
- Chaque code de récupération est à usage unique (Supabase gère)

API Supabase à utiliser :
- supabase.auth.mfa.enroll() pour le QR code
- supabase.auth.mfa.challenge() pour initier la vérification
- supabase.auth.mfa.verify() pour valider le code TOTP

Ne pas toucher au flux d'authentification des non-admins.
```

SEO & Référencement naturel

React est une SPA par défaut — ce qui signifie que le SEO n'est pas automatique. Voici comment structurer votre app Lovable pour être indexée efficacement par Google.

> ⚠️ **Lovable génère une SPA React — limitation SEO native**
>
> Par défaut, Google peut indexer les SPAs, mais avec un délai. Pour un SEO optimal sur un site vitrine ou un blog, préférez un rendu côté serveur (SSR) ou une génération statique (SSG). Pour une app SaaS, le SEO porte surtout sur la landing page publique.

Prompt — SEO de base sur toutes les pages

PROMPT — BALISES MÉTA ET OPEN GRAPH

```
Ajoute la gestion SEO complète via react-helmet-async.

Installe : react-helmet-async

Crée un composant SEOHead réutilisable avec ces props :
  title: string (obligatoire)
  description: string (150-160 chars)
  canonical?: string (URL canonique complète)
  ogImage?: string (URL image 1200x630px)
  noIndex?: boolean (true pour les pages privées)
  schema?: object (JSON-LD Schema.org)

Balises à générer dans SEOHead :









  {noIndex && }
  {schema && }

Utilisation sur chaque page :


Pages privées (dashboard, profil, admin) : noIndex={true}
Pages publiques : toutes les balises remplies avec contenu réel
```

Prompt — Sitemap XML automatique

PROMPT — GÉNÉRATION DU SITEMAP.XML

```
Crée un sitemap.xml dynamique pour le référencement.

Via une Supabase Edge Function GET /sitemap.xml :

1. Pages statiques (priorité haute) :
   / → priority 1.0, changefreq weekly
   /projets → priority 0.8, changefreq weekly
   /blog → priority 0.8, changefreq daily
   /contact → priority 0.6, changefreq monthly
   /a-propos → priority 0.5, changefreq monthly

2. Pages dynamiques (générées depuis Supabase) :
   /projets/:slug → SELECT slug, updated_at FROM projects
                    WHERE published = true
                    priority 0.7, changefreq monthly
   /blog/:slug → SELECT slug, published_at FROM posts
                 WHERE published_at IS NOT NULL
                 priority 0.6, changefreq never

Format de sortie :



    https://monsite.com/
    2026-01-15
    weekly
    1.0



Headers : Content-Type: application/xml
Accessible sans authentification (SELECT public sur tables)
```

Prompt — Schema.org JSON-LD

PROMPT — DONNÉES STRUCTURÉES SCHEMA.ORG

```
Ajoute les données structurées Schema.org sur les pages clés :

PAGE D'ACCUEIL — Organization :
{
  "@context": "https://schema.org",
  "@type": "LocalBusiness",
  "name": "[Nom entreprise]",
  "url": "https://[domaine]",
  "telephone": "[téléphone]",
  "address": {
    "@type": "PostalAddress",
    "streetAddress": "[adresse]",
    "addressLocality": "[ville]",
    "postalCode": "[code postal]",
    "addressCountry": "FR"
  },
  "openingHours": "Mo-Fr 09:00-18:00"
}

PAGE ARTICLE DE BLOG — Article :
{
  "@context": "https://schema.org",
  "@type": "Article",
  "headline": "[titre article]",
  "datePublished": "[published_at ISO]",
  "dateModified": "[updated_at ISO]",
  "author": { "@type": "Person", "name": "[auteur]" },
  "image": "[cover_image_url]",
  "description": "[excerpt]"
}

PAGE FAQ (si applicable) — FAQPage :
{
  "@context": "https://schema.org",
  "@type": "FAQPage",
  "mainEntity": [
    { "@type": "Question",
      "name": "[question]",
      "acceptedAnswer": { "@type": "Answer", "text": "[réponse]" }
    }
  ]
}

Intégrer via le composant SEOHead avec la prop schema={}.
```

Checklist SEO — Avant mise en ligne

🔍 SEO — Checklist complète

- [ ] **Balises title uniques** sur chaque page (50-60 caractères)
- [ ] **Meta descriptions uniques** sur chaque page (150-160 caractères)
- [ ] **Balises canonical** sur toutes les pages pour éviter le contenu dupliqué
- [ ] **Open Graph** complet (og:title, og:description, og:image 1200×630)
- [ ] **noindex sur les pages privées** : dashboard, profil, admin, /auth/*
- [ ] **Sitemap.xml** soumis dans Google Search Console
- [ ] **robots.txt** configuré (bloquer /admin, /api, /auth)
- [ ] **Images en WebP** avec attribut alt descriptif sur toutes les images
- [ ] **Core Web Vitals** : LCP < 2.5s, FID < 100ms, CLS < 0.1 (tester avec PageSpeed)
- [ ] **HTTPS actif** et redirection HTTP → HTTPS automatique
- [ ] **URLs propres** : /blog/mon-article-titre (kebab-case, pas d'ID numériques)
- [ ] **Schema.org JSON-LD** sur les pages clés (valider avec Rich Results Test Google)

Prompt — robots.txt

PROMPT — ROBOTS.TXT POUR LOVABLE

```
Crée un fichier public/robots.txt avec ce contenu :

User-agent: *
Allow: /
Allow: /blog/
Allow: /projets/
Disallow: /dashboard/
Disallow: /admin/
Disallow: /auth/
Disallow: /mon-compte/
Disallow: /api/

Sitemap: https://[votre-domaine]/sitemap.xml

Assure-toi que le fichier est servi statiquement
depuis le dossier public/ et accessible sans auth.
```

21 — STRIPE

Intégrer Stripe pas à pas

Stripe est l'intégration de paiement la plus courante avec Lovable. Voici le guide complet : paiements one-shot, abonnements, webhooks et portail client.

> 💳 **Architecture Stripe dans Lovable**
>
> La logique Stripe se passe toujours côté serveur (Supabase Edge Functions). La clé secrète Stripe ne doit jamais être dans le code frontend. Le frontend utilise uniquement la clé publique pour Stripe.js.

Étapes de configuration initiale

1. Créer un compte sur **stripe.com** et activer le mode test
2. Dans Stripe Dashboard → Developers → API Keys : copier la clé secrète test
3. Dans Lovable → Settings → Environment Variables : ajouter `STRIPE_SECRET_KEY`
4. Ajouter la clé publique dans le code frontend : `VITE_STRIPE_PUBLISHABLE_KEY`
5. Configurer le webhook endpoint (voir ci-dessous)

Prompt — Paiement one-shot (checkout session)

PROMPT — PAIEMENT UNIQUE AVEC STRIPE CHECKOUT

```
Implémente un paiement one-shot via Stripe Checkout.

1. EDGE FUNCTION create-checkout-session :
   - Reçoit : { amount_cents: number, product_name: string,
              user_id: string, metadata: object }
   - Vérifie le JWT de l'utilisateur
   - Crée une Stripe Checkout Session :
     mode: 'payment'
     success_url: https://[domaine]/paiement/succes?session_id={CHECKOUT_SESSION_ID}
     cancel_url: https://[domaine]/paiement/annule
     line_items: [{ price_data: { currency: 'eur',
       product_data: { name: product_name },
       unit_amount: amount_cents }, quantity: 1 }]
     metadata: { user_id, ...metadata }
   - Retourne { url: session.url }

2. FRONTEND — Bouton "Payer" :
   - Appelle l'Edge Function avec fetch
   - Redirige vers session.url (window.location.href = url)
   - État loading pendant la création de session

3. PAGE /paiement/succes :
   - Récupère session_id dans les query params
   - Appelle l'Edge Function verify-payment pour confirmer
   - Affiche la confirmation et met à jour le statut en DB

4. EDGE FUNCTION verify-payment :
   - stripe.checkout.sessions.retrieve(session_id)
   - Vérifie payment_status === 'paid'
   - Met à jour la DB en conséquence
   - Retourne { success: true, details }

Variables d'env nécessaires :
   STRIPE_SECRET_KEY (serveur uniquement)
   VITE_STRIPE_PUBLISHABLE_KEY (frontend)
```

Prompt — Abonnements récurrents

PROMPT — STRIPE SUBSCRIPTIONS

```
Implémente les abonnements récurrents Stripe.

Prérequis : créer les produits et prix dans Stripe Dashboard
  Plan Starter : price_XXXX (9.99€/mois)
  Plan Pro : price_YYYY (29.99€/mois)

1. TABLE subscriptions (colonnes contractuelles) :
   user_id: uuid (FK profiles), stripe_customer_id: text,
   stripe_subscription_id: text, plan: enum('starter','pro'),
   status: enum('active','past_due','cancelled','trialing'),
   current_period_end: timestamptz

2. EDGE FUNCTION create-subscription-checkout :
   - Crée ou récupère le Stripe Customer (via stripe_customer_id)
   - Crée Checkout Session mode='subscription'
   - price_id selon le plan choisi
   - trial_period_days: 14 (période d'essai gratuite)
   - success_url + cancel_url

3. EDGE FUNCTION stripe-webhook :
   - Endpoint : /functions/v1/stripe-webhook
   - Vérifie la signature Stripe (STRIPE_WEBHOOK_SECRET)
   - Gère les événements :
     customer.subscription.created → INSERT subscriptions
     customer.subscription.updated → UPDATE subscriptions
     customer.subscription.deleted → status='cancelled'
     invoice.payment_failed → status='past_due' + email alerte

4. PAGE de gestion d'abonnement /mon-abonnement :
   - Plan actuel + date de renouvellement
   - Bouton "Gérer mon abonnement" → Stripe Customer Portal
   - Bouton "Changer de plan"

5. EDGE FUNCTION create-portal-session :
   - stripe.billingPortal.sessions.create()
   - return_url: https://[domaine]/mon-abonnement
   - Redirige vers le portail Stripe (annulation, changement CB)
```

Prompt — Configuration du webhook Stripe

PROMPT — WEBHOOK STRIPE SÉCURISÉ

```
Configure le webhook Stripe de manière sécurisée.

Dans Stripe Dashboard → Webhooks → Add endpoint :
  URL : https://[projet].supabase.co/functions/v1/stripe-webhook
  Événements à écouter :
    payment_intent.succeeded
    payment_intent.payment_failed
    customer.subscription.created
    customer.subscription.updated
    customer.subscription.deleted
    invoice.payment_succeeded
    invoice.payment_failed

Dans Supabase Edge Function stripe-webhook :
  1. Lire la signature : request.headers.get('stripe-signature')
  2. Vérifier : stripe.webhooks.constructEvent(
       body, signature, Deno.env.get('STRIPE_WEBHOOK_SECRET'))
  3. Si erreur de signature → retourner 400 immédiatement
  4. Traiter l'événement selon event.type
  5. Retourner 200 dans tous les cas de succès
     (Stripe retente si 5xx ou timeout)

Variable d'env : STRIPE_WEBHOOK_SECRET
(disponible dans Stripe Dashboard → Webhook → Signing secret)

IMPORTANT :
- Toujours retourner 200 même si l'événement n'est pas géré
- Idempotence : vérifier si l'événement a déjà été traité
  (stocker stripe_event_id en DB pour éviter les doublons)
```

Cartes de test Stripe

Numéro de carte	Résultat	Usage
`4242 4242 4242 4242`	✓ Paiement réussi	Test nominal
`4000 0000 0000 0002`	✗ Carte refusée	Test d'échec
`4000 0025 0000 3155`	⚡ 3D Secure requis	Test authentification forte
`4000 0000 0000 9995`	✗ Fonds insuffisants	Test d'erreur métier

💡 **Passer en mode live**

Remplacez simplement les clés de test par les clés live dans vos variables d'environnement de production. Ne jamais mélanger les clés test et live dans le même environnement.

---

22 — INTÉGRATION D'APIS

Intégrer des APIs externes

Lovable peut consommer n'importe quelle API REST ou GraphQL. Les appels sensibles passent par les Edge Functions Supabase pour protéger vos clés. Voici la méthode complète.

Principe architectural des appels API

Type d'appel	Depuis	Pourquoi
APIs publiques sans clé (météo, taux de change…)	Frontend React directement	Pas de secret à protéger
APIs avec clé secrète (OpenAI, Stripe, Twilio…)	Supabase Edge Function	Clé jamais exposée côté client
APIs OAuth utilisateur (Google, GitHub…)	Supabase Auth + Edge Function	Token utilisateur côté serveur
Webhooks entrants (Stripe, Make, GitHub…)	Supabase Edge Function	Endpoint public sécurisé

Prompt — Appel API REST publique (côté client)

API PUBLIQUE SANS CLÉ

```
Intègre l'API [Nom] (endpoint : GET https://api.exemple.com/v1/ressource) dans [NomComposant].

Comportement :
- Appel au montage (useEffect, dépendances [param])
- État loading : skeleton loader pendant la requête
- État erreur : message + bouton "Réessayer" (retry avec backoff exponentiel)
- État vide : illustration + message descriptif
- Cache : stocker en state local, re-fetcher si paramètre change

Ne pas passer par une Edge Function (API publique sans clé secrète).
Ne pas stocker en Supabase sauf besoin explicite.
```

Prompt — API avec clé secrète via Edge Function

API SÉCURISÉE — PATTERN GÉNÉRIQUE RÉUTILISABLE

```
Crée une Edge Function call-[nom-api] pour appeler [Nom API].

1. Vérifier le JWT :
   const { data: { user } } = await supabase.auth.getUser(token)
   Si pas d'utilisateur → retourner 401

2. Valider les paramètres entrants (ne jamais faire confiance au client)

3. Appel API avec clé secrète :
   const response = await fetch('https://api.exemple.com/endpoint', {
     method: 'POST',
     headers: {
       'Authorization': `Bearer ${Deno.env.get('API_SECRET_KEY')}`,
       'Content-Type': 'application/json'
     },
     body: JSON.stringify({ ...params })
   })

4. Si response.ok === false → logger + retourner erreur générique
   (ne jamais exposer les détails de l'erreur API au client)

5. Retourner uniquement les données nécessaires au frontend

Variable d'env : API_SECRET_KEY
Frontend : supabase.functions.invoke('call-[nom-api]', { body: params })
```

Prompt — OpenAI GPT avec streaming

EDGE FUNCTION OPENAI GPT-4O AVEC STREAMING SSE

```
Edge Function openai-chat :

1. Vérifier JWT utilisateur
2. Rate limiting : max 10 appels/heure/utilisateur
   (table api_rate_limits : user_id, endpoint, count, reset_at)
3. Appel OpenAI avec streaming :
   const stream = await openai.chat.completions.create({
     model: 'gpt-4o',
     messages: [
       { role: 'system', content: Deno.env.get('SYSTEM_PROMPT') },
       { role: 'user', content: userMessage }
     ],
     stream: true,
     max_tokens: 1000
   })
4. Retourner le stream au frontend (Server-Sent Events)

Frontend — composant AIChat :
- Input + bouton Envoyer + bouton "Arrêter"
- Affichage stream token par token
- Historique en state local (pas en DB pour le MVP)

Variables d'env : OPENAI_API_KEY, SYSTEM_PROMPT
```

Prompt — API GraphQL avec Apollo Client

GRAPHQL DEPUIS LOVABLE

```
Intègre l'API GraphQL [Nom] (endpoint : https://api.exemple.com/graphql).
Installe : @apollo/client

Configuration dans src/lib/apollo.ts :
const client = new ApolloClient({
  uri: 'https://api.exemple.com/graphql',
  cache: new InMemoryCache(),
  headers: { 'Authorization': `Bearer ${token}` }
})

Query à implémenter :
query GetItems($filter: String, $limit: Int) {
  items(filter: $filter, limit: $limit) {
    id, name, description, createdAt
  }
}

Dans [NomComposant] :
const { data, loading, error } = useQuery(GET_ITEMS, {
  variables: { filter, limit: 20 }
})

Si la clé est secrète : proxy Edge Function qui fait la requête GraphQL côté serveur.
```

Prompt — Webhooks entrants sécurisés avec idempotence

ENDPOINT WEBHOOK GÉNÉRIQUE SÉCURISÉ

```
Edge Function webhook-[service] :

1. Vérifier la signature :
   const sig = req.headers.get('[service]-signature')
   const isValid = await verifySignature(body, sig, Deno.env.get('WEBHOOK_SECRET'))
   Si invalide → retourner 401 immédiatement

2. Idempotence : vérifier si event_id déjà traité
   SELECT id FROM processed_webhooks WHERE event_id = event.id
   Si déjà traité → retourner 200 sans retraiter

3. Router selon event.type :
   switch(event.type) {
     case 'event.created': await handleCreated(event.data); break;
     case 'event.updated': await handleUpdated(event.data); break;
     default: console.log('Unhandled:', event.type);
   }

4. Logger dans webhook_logs : event_id, type, status, payload, processed_at

5. Toujours retourner 200 pour les événements connus
   (retourner 5xx déclenche les retentatives du service)

Variable d'env : WEBHOOK_SECRET
```

Prompt — Retry avec backoff exponentiel

UTILITAIRE FETCHWITHRETRY ROBUSTE

```
async function fetchWithRetry(url, options, maxRetries = 3) {
  for (let attempt = 0; attempt <= maxRetries; attempt++) {
    try {
      const response = await fetch(url, options)

      // Retenter uniquement sur 5xx et 429 (rate limiting)
      if (response.status === 429 || response.status >= 500) {
        if (attempt === maxRetries) throw new Error(`Failed after ${maxRetries} retries`)
        // Backoff exponentiel + jitter aléatoire
        const delay = Math.pow(2, attempt) * 1000 + Math.random() * 500
        await new Promise(r => setTimeout(r, delay))
        continue
      }
      return response // 2xx et 4xx : retourner immédiatement
    } catch (err) {
      if (attempt === maxRetries) throw err
      await new Promise(r => setTimeout(r, Math.pow(2, attempt) * 1000))
    }
  }
}

Ne jamais retenter les 4xx (sauf 429) — ce sont des erreurs côté client.
Utiliser dans toutes les Edge Functions appelant des APIs critiques.
```

Checklist intégration API

🔌 Avant de déployer une intégration API

- [ ] Clé API en variable d'environnement — jamais dans le code ou commité
- [ ] Appels avec clé secrète passent par une Edge Function (jamais côté client)
- [ ] Vérification JWT dans chaque Edge Function avant l'appel API
- [ ] Rate limiting sur les APIs coûteuses (OpenAI, IA génération…)
- [ ] Erreurs API renvoyées en messages génériques (pas de stack trace au client)
- [ ] Retry avec backoff sur les erreurs 5xx et 429
- [ ] Idempotence pour les webhooks (event_id stocké pour éviter les doublons)
- [ ] Tous les états UI : loading, erreur, vide gérés dans les composants
- [ ] Test en mode sandbox/test avant activation du mode production

23 — EXTENSIONS & INTÉGRATIONS

Extensions & intégrations disponibles

Lovable s'intègre avec de nombreux services externes via des APIs, des webhooks ou des intégrations natives. Voici les plus utiles, organisées par catégorie.

📊 Analytics & tracking

GA4 **Google Analytics 4**

Prompt : "Intègre Google Analytics 4 avec mesure des events clés (signup, checkout, cta_click). Utilise gtag.js via le composant Helmet."

PLB **Plausible Analytics**

Alternative privacy-first à GA4, RGPD par défaut, pas de cookies. 1 ligne de script. Idéal pour les apps européennes.

MXP **Mixpanel**

Analytics produit avancé : funnels, rétention, cohortes. Prompt : "Intègre Mixpanel et track les events signup, first_action, conversion."

HJR **Hotjar / Microsoft Clarity**

Heatmaps et enregistrements de sessions. Clarity est gratuit et illimité. Utile pour débugger l'UX sans effort.

📧 Emails transactionnels

RSN
Resend (recommandé)
API email moderne, intégration native Supabase. Prompt : "Configure Resend pour les emails transactionnels via Supabase Edge Function. Template : confirmation d'inscription."

SG
SendGrid
Leader historique. Free tier 100 emails/jour. Bien documenté pour les Edge Functions Supabase. Templates drag-and-drop dans le dashboard.

MG
Mailgun
Excellente délivrabilité pour les emails B2B. API simple, SDK Node.js utilisable dans les Edge Functions Deno.

LPS
Loops
Email marketing + transactionnel pour SaaS. Segments automatiques basés sur les events. Idéal pour les onboarding sequences.

🤖 Intelligence artificielle

OAI
OpenAI API
Prompt : "Crée une Edge Function qui appelle OpenAI GPT-4o pour [cas d'usage]. Clé en OPENAI_API_KEY. Retourne du streaming JSON."

ANT
Anthropic Claude API
API de Claude directement dans votre app. Même pattern que OpenAI via Edge Function. Clé en ANTHROPIC_API_KEY.

PGV
pgvector (Supabase)
Extension PostgreSQL pour la recherche vectorielle (RAG, similarité sémantique). Activable directement dans Supabase. Idéal pour les chatbots sur vos données.

RPC
Replicate
Générer des images (Stable Diffusion, FLUX) via API. Prompt : "Intègre Replicate pour générer une image depuis un prompt utilisateur dans l'Edge Function generate-image."

🔔 Notifications & communication

TWL
Twilio SMS
Envoi de SMS depuis une Edge Function. Utilisé pour les OTP personnalisés, rappels de RDV, alertes critiques. API très simple.

SLK
Slack Webhooks
Notifier votre équipe Slack d'un événement (nouveau signup, erreur critique, paiement). Webhook entrant en 2 minutes.

PH
Push notifications (OneSignal)
Notifications push web sur desktop et mobile. SDK JavaScript + Edge Function pour déclencher. Plan gratuit généreux.

ISB
Supabase Realtime
Natif à Supabase. Notifications en temps réel dans l'app via WebSocket. Prompt : "Abonne le composant NotifBell aux changements de la table notifications via Supabase Realtime."

🔗 Automatisation & no-code

MK

Make.com (ex-Integromat)

Connectez Lovable à 1000+ apps sans code. Utilisez un webhook Supabase Edge Function pour déclencher des scénarios Make (CRM, Notion, Airtable…).

N8N

n8n (self-hosted)

Alternative open-source à Make. Auto-hébergeable. Idéal pour les projets qui veulent garder le contrôle des automatisations et des données.

ZAP

Zapier

Le plus connu des automatiseurs. Connectez Supabase Webhooks à Zapier pour synchroniser avec Gmail, HubSpot, Salesforce, etc.

ART

Airtable API

Synchronisez des données Lovable/Supabase avec une base Airtable. Utile quand une équipe non-tech gère des données dans Airtable.

📁 Stockage & fichiers

SBS

Supabase Storage (natif)

Stockage de fichiers intégré. Prompt : "Configure un bucket Supabase Storage 'avatars' avec upload depuis le profil utilisateur. Limite : 5MB, formats jpg/png/webp."

CDY

Cloudinary

Optimisation et transformation d'images à la volée (resize, crop, WebP auto). Idéal pour les galeries photos. Free tier 25GB.

S3

AWS S3 / Cloudflare R2

Pour les gros volumes de fichiers. Cloudflare R2 est S3-compatible et sans frais de sortie. Configuration via Edge Function.

UPD

UploadThing

Upload de fichiers moderne, conçu pour les apps React. SDK simple, gestion des types de fichiers, limits et callbacks. Intégration en 30 min.

🗺️ Cartes & géolocalisation

MBX

Mapbox

Cartes interactives personnalisables. Prompt : "Intègre Mapbox GL JS pour afficher les [points de la table locations] sur une carte avec des markers cliquables."

GMP

Google Maps API

Autocomplete d'adresses, géocodage, calcul d'itinéraires. Clé en variable d'environnement. Free tier 200$/mois de crédit.

Tableau récapitulatif des intégrations

Service	Catégorie	Complexité	Free tier	Intégration via
Stripe	Paiement	Moyenne	Test uniquement	Edge Function
Resend	Email	Facile	100/jour	Edge Function
OpenAI	IA	Moyenne	Non	Edge Function
Google Analytics 4	Analytics	Facile	Oui	Script JS (Helmet)
Plausible	Analytics	Très facile	Payant (9$/mois)	Script JS
Make.com	Automatisation	Facile	1000 ops/mois	Webhook
Twilio	SMS	Moyenne	Crédits test	Edge Function
Supabase Realtime	Temps réel	Facile	Inclus Supabase	SDK natif
Cloudinary	Médias	Facile	25GB	SDK JS direct
Mapbox	Cartes	Moyenne	50k req/mois	SDK JS direct

Prompts d'intégration rapide

PROMPT — INTÉGRATION MAKE.COM VIA WEBHOOK

```
Quand un utilisateur remplit le formulaire de contact,
déclenche un webhook vers Make.com pour automatiser
la qualification du lead.

1. Crée une Edge Function send-to-make :
   - Reçoit les données du formulaire
   - Fait un POST vers MAKE_WEBHOOK_URL (variable d'env)
   - Payload JSON avec tous les champs du formulaire
   - Gère les erreurs (ne pas bloquer la soumission si Make échoue)

2. Dans le formulaire de contact :
   - Appelle send-to-make en parallèle de l'INSERT Supabase
   - Le résultat Make n'est pas attendu (fire-and-forget)

Variable d'env : MAKE_WEBHOOK_URL
(disponible dans Make.com → Webhooks → Custom webhook → URL)
```

PROMPT — UPLOAD DE FICHIERS AVEC SUPABASE STORAGE

```
Implémente l'upload d'avatar sur la page profil.

1. Configuration Supabase Storage :
   Bucket 'avatars' : public, limité à jpg/png/webp, max 5MB

2. Composant AvatarUpload :
   - Bouton "Changer ma photo" avec input file caché
   - Preview de l'image avant upload
   - Barre de progression pendant l'upload
   - Compression côté client avant upload (browser-image-compression)
   - Nom du fichier : {user_id}/avatar.{ext}

3. Après upload :
   UPDATE profiles SET avatar_url = [URL publique Supabase Storage]
   WHERE id = auth.uid()

4. Politique Storage RLS :
   SELECT : public (les avatars sont visibles par tous)
   INSERT/UPDATE/DELETE : auth.uid()::text = (storage.foldername(name))[1]

Gestion d'erreur :
   - Fichier trop lourd : "Fichier trop volumineux (max 5MB)"
   - Format non supporté : "Format non supporté (JPG, PNG, WebP)"
   - Erreur réseau : "Erreur d'upload, veuillez réessayer"
```

PROMPT — INTÉGRATION GOOGLE ANALYTICS 4

```
Intègre Google Analytics 4 dans l'application.

1. Ajouter dans index.html (via react-helmet-async dans App.tsx) :
   Script gtag.js avec MEASUREMENT_ID depuis variable d'env
   (VITE_GA_MEASUREMENT_ID)

2. Créer un hook useAnalytics() avec ces méthodes :
   trackPageView(path: string)
   trackEvent(name: string, params: object)
   trackConversion(value: number, currency: string)

3. Déclencher automatiquement :
   trackPageView() à chaque changement de route (React Router)

4. Events à tracker manuellement :
   trackEvent('sign_up', { method: 'email'|'google'|'github' })
   trackEvent('login', { method: 'email'|'google' })
   trackEvent('contact_form_submit', { project_type, budget })
   trackEvent('checkout_started', { plan, amount })
   trackEvent('purchase', { transaction_id, value, currency })

5. Pages privées (/dashboard, /admin) :
   Anonymiser les données : ne pas tracker les actions internes

RGPD : afficher une bannière cookies (consentement requis en EU)
avant d'initialiser gtag.
Variable d'env : VITE_GA_MEASUREMENT_ID=G-XXXXXXXXXX
```

BONUS — BIBLIOTHÈQUE DE PROMPTS

30 prompts avancés prêts à l'emploi

Prompts de référence pour les situations les plus courantes — copiez, adaptez, itérez.

Performance & optimisation

PROMPT — OPTIMISATION CORE WEB VITALS

```
Optimise les performances de l'application pour améliorer
les Core Web Vitals sans toucher à la logique métier.

Actions à effectuer :
1. Code splitting : ajouter React.lazy() et Suspense sur
   toutes les routes sauf la page d'accueil
2. Images : remplacer les balises  par un composant
   LazyImage avec loading="lazy" et format WebP
3. Fonts : supprimer les imports Google Fonts, utiliser
   les system fonts (font-family: system-ui, sans-serif)
4. Bundle size : auditer les imports et remplacer les imports
   globaux par des imports ciblés (ex: import { Button }
   from '@shadcn/ui' plutôt que l'index)
5. Memoization : identifier les composants qui re-rendent
   inutilement et ajouter React.memo() où pertinent

Ne pas modifier :
- La logique métier ou les API calls
- Les styles existants
- La structure des routes

Mesure attendue : rapport console.log de la taille du bundle
avant et après.
```

PROMPT — PAGINATION INFINIE

```
Implémente la pagination infinie sur la liste [NomListe].

Comportement :
- Charger 20 éléments au premier rendu
- Détecter le scroll vers le bas (Intersection Observer)
- Charger 20 éléments supplémentaires automatiquement
- Indicateur "Chargement..." pendant le fetch
- Message "Tous les éléments sont chargés" en fin de liste
- Pas de bouton "Charger plus" (scroll infini uniquement)

Requête Supabase :
  .from('table').select('*')
  .range(page * 20, (page + 1) * 20 - 1)
  .order('created_at', { ascending: false })

État local :
  items: [] (liste cumulative)
  page: 0 (page courante)
  hasMore: true (encore des éléments à charger)
  loading: false

Ne pas refactoriser les composants existants.
Ne modifier que le composant [NomListe].
```

PROMPT — RECHERCHE TEMPS RÉEL AVEC DEBOUNCE

```
Ajoute une recherche en temps réel sur [NomPage].

Comportement :
- Input de recherche avec placeholder "Rechercher..."
- Debounce de 300ms avant la requête Supabase
- Recherche dans les colonnes : [col1], [col2], [col3]
  (utiliser ilike '%terme%' pour chaque colonne)
- Highlight du terme recherché dans les résultats
- Compteur "X résultat(s) trouvé(s)"
- Bouton clear (×) si le champ n'est pas vide
- État vide : "Aucun résultat pour '[terme]'"

Requête Supabase :
  .or('col1.ilike.%terme%,col2.ilike.%terme%')

Performance :
- Annuler la requête précédente si une nouvelle est lancée
  (AbortController)
- Ne pas rechercher si moins de 2 caractères

Ne modifier que le composant [NomPage].
Ne pas toucher aux autres composants.
```

PROMPT — EXPORT CSV DES DONNÉES

```
Ajoute un bouton "Exporter CSV" sur la page [NomPage].

Comportement :
- Bouton "Exporter CSV" avec icône de téléchargement
- Récupère TOUTES les données filtrées (pas seulement la page courante)
- Colonnes à exporter : [col1: "Libellé 1"], [col2: "Libellé 2"]...
- Encodage UTF-8 avec BOM pour compatibilité Excel
- Nom du fichier : [nom-export]-YYYY-MM-DD.csv
- Toast de confirmation après téléchargement

Code de génération CSV :
  const BOM = '\uFEFF'
  const headers = ['Libellé 1', 'Libellé 2', ...]
  const rows = data.map(item => [item.col1, item.col2, ...])
  const csv = BOM + [headers, ...rows]
    .map(row => row.map(cell =>
      `"${String(cell ?? '').replace(/"/g, '""')}"`
    ).join(',')).join('\n')

RLS : utiliser les mêmes filtres que la vue courante.
Admin uniquement si la table contient des données sensibles.
```

Internationalisation & accessibilité

PROMPT — MODE SOMBRE (DARK MODE)

```
Implémente le dark mode avec Tailwind CSS.

1. Configuration tailwind.config.js :
   darkMode: 'class' (basé sur la classe 'dark' sur )

2. Context ThemeProvider :
   - Lire la préférence système : window.matchMedia('prefers-color-scheme: dark')
   - Persister le choix en localStorage ('theme': 'light'|'dark'|'system')
   - Appliquer la classe 'dark' sur document.documentElement

3. Toggle dans le header :
   - Icône soleil/lune selon le thème actuel
   - 3 options dans un dropdown : Clair / Sombre / Système

4. Mise à jour des composants :
   - Remplacer les couleurs hardcodées par des variables Tailwind
     dark-aware : bg-white dark:bg-gray-900
   - Vérifier le contraste des textes en dark mode (WCAG AA)

Ne pas modifier la logique métier.
Tester visuellement sur les pages principales avant de valider.
```

PROMPT — ACCESSIBILITÉ WCAG AA

```
Améliore l'accessibilité de l'application (niveau WCAG AA).

Actions à effectuer :
1. Images : ajouter alt descriptif sur toutes les
   (alt="" pour les images décoratives)
2. Formulaires : associer chaque  à son
                                  

   via htmlFor/id
3. Boutons : s'assurer qu'aucun bouton n'a uniquement
   une icône sans aria-label
4. Navigation clavier : vérifier que Tab/Shift+Tab
   navigue dans l'ordre logique
5. Couleurs : vérifier ratio de contraste > 4.5:1 pour le texte
6. Focus visible : ne pas supprimer outline sans alternative
7. Titres : hiérarchie H1 → H2 → H3 respectée sur chaque page
8. Skip link : ajouter "Aller au contenu principal" en premier
   élément focusable de chaque page

Tester avec : axe DevTools (extension Chrome, gratuite)
Ne modifier que le HTML/ARIA, pas la logique.
```

Internationalisation

PROMPT — I18N FRANÇAIS / ANGLAIS

```
Ajoute le support multilingue FR/EN avec react-i18next.

1. Installation : react-i18next, i18next, i18next-browser-languagedetector

2. Structure des fichiers :
   src/locales/fr/common.json → textes français
   src/locales/en/common.json → textes anglais

3. Configuration i18n.ts :
   - Détection automatique de la langue du navigateur
   - Fallback sur 'fr' si langue non supportée
   - Persistance du choix en localStorage

4. Migration des textes hardcodés :
   - Remplacer tous les textes UI par t('clé')
   - Conserver les données de la DB telles quelles
     (pas de traduction des données Supabase)

5. Toggle de langue dans le header :
   Drapeaux FR / EN, persisté en localStorage

Commencer par les composants :
   Header, Footer, LoginPage, SignupPage
Ne pas migrer les pages admin dans cette étape.
```