JETC_Immo – Guide d ’ instructions pour IA (mode A→Z)

JETC_Immo – Guide d ’ instructions pour IA (mode exécution stricte A→Z)
==========

1. RÔLE ET COMPORTEMENT DE L ’ IA

Tu es une IA de type développeur (Codex) chargée de construire ou refondre l ’ application JETC_Immo.

Règles de comportement obligatoires :
1. Tu dois suivre les étapes de ce document strictement dans l ’ ordre, de l ’ étape 1 à l ’ étape finale.
2. Tu ne passes JAMAIS à l ’ étape suivante tant que :
- le code de l ’ étape en cours n ’ est pas cohérent
- les vérifications indiquées pour cette étape ne sont pas satisfaites.
3. À chaque étape :
- tu expliques ce que tu vas faire
- tu proposes le code nécessaire (backend / SQL / frontend selon l ’ étape)
- tu indiques comment tester le résultat
4. Si une erreur, une incohérence ou une ambiguïté est détectée :
- tu t ’ arrêtes immédiatement
- tu expliques le problème
- tu proposes des pistes de correction
- tu attends une confirmation de l ’ utilisateur avant de continuer.
5. Tu écris un code propre, commenté, structuré, en privilégiant :
- lisibilité
- robustesse
- séparation claire des responsabilités.
6. Tu assumes que la base de données est sous Supabase (PostgreSQL + Auth + RLS + Storage).
7. Tu indiques clairement les variables d ’ environnement nécessaires et leur rôle.
8. Tu maintiens la cohérence avec la structure métier décrite dans ce document.

Objectif global :
Construire une application JETC_Immo permettant de gérer les relations entre :
- Régies
- Propriétaires
- Immeubles et logements
- Locataires
- Entreprises
- Techniciens
- Tickets d ’ intervention
- Missions
- Facturation
- Messagerie

- Notifications

avec un niveau de sécurité compatible avec RLS Supabase.

2. CONTEXTE MÉTIER

Résumé du domaine :

1. La régie gère un portefeuille d ' immeubles et de logements.
2. Chaque logement peut avoir un ou plusieurs locataires au cours du temps.
3. Les locataires déclarent des problèmes via des tickets (plomberie, électricité, serrurerie, etc.).
4. La régie valide et diffuse les tickets à des entreprises partenaires.
- Mode général : toutes les entreprises éligibles peuvent voir le ticket.
- Mode restreint : seules les entreprises sélectionnées voient le ticket.
5. Une entreprise accepte une intervention, ce qui crée une mission.
6. Un technicien est assigné à la mission.
7. Le technicien intervient, peut signaler du retard, remplir un rapport, et faire signer le locataire.
8. La mission génère une facture, avec une commission pour JETC_Immo.
9. Le système conserve un historique des interventions, échanges (chat) et notifications.

Rôles principaux :
- Locataire
- Régie
- Entreprise
- Technicien
- Propriétaire (consultation)
- Administrateur (optionnel)

3. ARCHITECTURE GLOBALE

Stack recommandée :

- Backend : Supabase (PostgreSQL + Auth + RLS) + API routes (par exemple sur Vercel).
- Frontend : application web (HTML/JS ou framework type Next.js).
- Auth : Supabase Auth (email/password).
- Stockage fichiers : Supabase Storage (photos de dégâts, signatures).

Principe :
- Le frontend ne contacte jamais la base directement avec des droits larges.
- Les opérations sensibles passent par des endpoints backend sécurisés.
- Les règles RLS empêchent l ' accès illégitime aux données même en cas de fuite de clé.

4. SCHÉMA DE DONNÉES (RÉSUMÉ FONCTIONNEL)

Le schéma détaillé peut déjà être créé dans Supabase, mais l ' IA doit respecter la logique suivante :

Principales entités :

1. Régies (regies)
- id, nom, email_contact, téléphone, adresse
- options : astreinte, diffusion_mode (general/restreint), entreprises_autorisees

2. Propriétaires (proprietaires)
- rattachés à une régie
- informations de contact

3. Immeubles (immeubles)
- rattachés à une régie
- adresse, ville, NPA, etc.

4. Logements (logements)
- rattachés à un immeuble
- informations spécifiques (numéro, digicode, accès, caractéristiques)

5. Locataires (locataires)
- liés à auth.users
- liés à un logement

6. Profils (profiles)
- un par utilisateur auth.users
- rôle (locataire, regie, entreprise, technicien, admin)
- regie_id, entreprise_id si applicable
- paramètres de notification

7. Entreprises (entreprises)
- liées à une régie
- coordonnées, tarifs, zones couvertes, astreinte

8. Techniciens (techniciens)
- liés à une entreprise
- liés à profiles
- compétences, disponibilité

9. Tickets (tickets)
- locataire, logement, régie
- catégorie, sous_catégorie, description
- disponibilités du locataire
- mode de diffusion (general / restreint)
- budget (plafond, estimation)
- statut, dates

10. Missions (missions)
- liées à un ticket
- entreprise, technicien, régie, locataire
- dates (acceptation, intervention, fin)

- statut, signature, rapports

11. Matériel (materiel_categories, materiel_items, mission_materiel)
- matériel par entreprise
- matériel utilisé sur une mission

12. Factures (factures)
- mission, entreprise, montant, commission JETC, statut

13. Messages (messages)
- chat entre acteurs sur une mission donnée

14. Notifications (notifications)
- notifications générées automatiquement (nouveau ticket, mission acceptée, retard, etc.).

5. RÈGLES DE SÉCURITÉ (RLS) – PRINCIPES

L ' IA doit implémenter des règles RLS compatibles avec les principes suivants :

1. Un locataire :
- ne voit que ses propres tickets et ses propres données.
2. Une régie :
- ne voit que les données (logements, locataires, tickets, missions) de sa régie.
3. Une entreprise :
- voit les tickets qui lui sont accessibles :
• en mode général, selon réglages régie,
• en mode restreint, si explicitement autorisée.
- voit ses missions et ses techniciens.
4. Un technicien :
- voit uniquement ses missions.
5. Le public :
- ne voit pas de données sensibles (optionnellement, un niveau ultra restreint pour un " catalogue " de tickets anonymisés).
6. Les accès Storage (photos, signatures) :
- doivent être limités aux acteurs concernés par la mission.

6. PLAN DE DÉVELOPPEMENT POUR L ' IA (ÉTAPES A→Z)

Les étapes suivantes doivent être suivies STRICTEMENT dans l ' ordre donné.

À chaque étape :
- L ' IA décrit ce qu ' elle fait.
- L ' IA propose le code.
- L ' IA explique comment tester.
- L ' IA s ' arrête si une incohérence est détectée.

Étape 1 – Initialisation du projet

Objectif :
- Définir la structure du projet (backend + frontend minimal).
- Configurer les variables d ' environnement pour Supabase.

Attendu :
- Arborescence de base du projet.
- Fichiers de configuration minimal (par ex. .env.example).
- Documentation rapide sur comment lancer le projet en local.

Vérifications :
- Arborescence cohérente.
- Place prévue pour les routes API.
- Place prévue pour le frontend.

Étape 2 – Connexion Supabase & Auth

Objectif :
- Connecter le backend à Supabase.
- Mettre en place l ' authentification email/password.
- Implémenter une route de login qui renvoie l ' utilisateur connecté et son profil.

Attendu :
- Client Supabase côté serveur.
- Route /auth/login (ou équivalent) :
• prend email + mot de passe
• appelle Supabase Auth
• récupère profil associé (table profiles)
• renvoie : userId, role, regie_id, entreprise_id, etc.

Vérifications :
- Un utilisateur test peut se connecter.
- Une erreur claire est renvoyée si email/mot de passe est invalide.
- Les paramètres Supabase (URL, clés) sont externalisés dans des variables d ' environnement.

Étape 3 – Profils et rôles

Objectif :
- Garantir qu ' à la création d ' un utilisateur, un profil est créé dans la table profiles.

Attendu :
- Trigger SQL (ou logique backend) pour :
• créer un profil par défaut (role = locataire) à la création d ' un utilisateur.
- Route API pour récupérer le profil courant.

Vérifications :
- Création d ' un nouvel utilisateur -> création d ' un profil automatique.

- Récupération du rôle, regie_id, entreprise_id, is_active.

Étape 4 – Gestion régies / immeubles / logements / locataires
---
Objectif :
- Implémenter les opérations minimales pour :
• créer et maintenir les régies
• créer les immeubles
• créer les logements
• lier les locataires aux logements.

Attendu :
- Endpoints backend (réservés aux administrateurs/régies) pour :
• créer une régie
• créer un immeuble rattaché à une régie
• créer un logement rattaché à un immeuble
• associer un locataire (user) à un logement.

Vérifications :
- Les FK sont respectées.
- Un locataire test peut être lié à un logement.
- Une régie ne peut pas voir ou modifier les données d ' une autre régie (via RLS).

Étape 5 – Création de tickets

Objectif :
- Permettre à un locataire connecté de créer un ticket.

Attendu :
- Formulaire frontend pour :
• catégorie, sous_catégorie, description
• photos éventuelles (gérées plus tard)
• disponibilités (dispo1, dispo2, dispo3)
- Endpoint backend :
• crée un ticket lié au locataire, logement, régie.
• enregistre statut initial (ex : "nouveau").

Vérifications :
- Un locataire ne peut créer un ticket que pour son logement.
- La régie associée au logement est bien renseignée dans le ticket.
- Aucune fuite de données vers d ' autres locataires.

Étape 6 – Diffusion des tickets aux entreprises

Objectif :
- Implémenter le mode général / restreint.

Attendu :
- Champs diffusion_mode et entreprises_autorisees gérés.
- Endpoint pour la régie :
• choisir le mode de diffusion
• définir entreprises autorisées en mode restreint.
- RLS correspondantes pour que seules les entreprises autorisées voient les tickets.

Vérifications :
- Une entreprise non autorisée ne voit pas le ticket.
- En mode général, seules les entreprises de la régie voient les tickets concernés.

Étape 7 – Acceptation et création de mission

Objectif :
- Permettre à une entreprise d ' accepter un ticket et créer une mission.

Attendu :
- Endpoint backend :
• prend ticket_id
• vérifie droits de l ' entreprise
• crée une mission avec statut "en_attente" ou "planifiée".
- Logique pour empêcher deux entreprises de prendre le même ticket en doublon.

Vérifications :
- Un ticket accepté n ' est plus proposé aux autres entreprises (ou passe dans un statut adapté).
- La mission contient bien toutes les références (ticket, régie, locataire, entreprise).

Étape 8 – Gestion des techniciens et planning

Objectif :
- Permettre à l ' entreprise de gérer ses techniciens et leur assigner des missions.

Attendu :
- Endpoints backend pour :
• créer / lier techniciens à une entreprise
• assigner une mission à un technicien
• définir la date d ' intervention planifiée.

Vérifications :
- Un technicien ne voit que ses propres missions.
- Une entreprise ne peut pas assigner de mission à un technicien d ' une autre entreprise.

Étape 9 – Intervention, retard, rapport, signature

Objectif :
- Gérer le cycle de vie d ' une mission côté technicien.

Attendu :
- Endpoints backend pour :
• démarrer une mission
• signaler un retard (avec motif prédéfini)
• clôturer une mission
• enregistrer un rapport (texte, matériel utilisé)
• gérer la signature du locataire (lien vers fichier Storage ou équivalent).

Vérifications :
- La mission passe par des statuts cohérents (en_attente, en_cours, terminée, etc.).
- Le retard génère une notification au locataire.
- La signature est correctement liée à la mission.

Étape 10 – Facturation

Objectif :
- Générer des factures à partir des missions terminées.

Attendu :
- Table factures alimentée à la clôture de mission.
- Montant calculé (simple dans un premier temps).
- Commission JETC renseignée.
- Statut de facture (en_attente, payée).

Vérifications :
- Une facture est liée à une unique mission.
- Une entreprise ne voit que ses factures.
- Possibilité de lister les factures par régie (vision agrégée).

Étape 11 – Messagerie (chat)

Objectif :
- Permettre des échanges entre :
• locataire ↔ régie ↔ entreprise ↔ technicien.

Attendu :
- Table messages utilisée avec :
• mission_id
• sender_id (profile)
• receiver_id (profile ou groupe logique)
- Endpoints pour :
• envoyer un message
• lister les messages par mission.

Vérifications :
- Les messages ne sont accessibles qu ' aux acteurs impliqués dans la mission.

- Un utilisateur ne peut pas consulter le chat d ' une mission étrangère.

Étape 12 – Notifications

Objectif :
- Générer des notifications sur les événements importants :
• nouveau ticket
• ticket accepté
• mission planifiée
• retard signalé
• mission terminée
• facture créée.

Attendu :
- Fonctions backend ou triggers SQL pour insérer dans la table notifications.
- Endpoints pour lister les notifications non lues et les marquer comme lues.

Vérifications :
- Chaque événement clé génère bien une notification pour les acteurs concernés.
- Les notifications respectent les RLS (un utilisateur ne voit que les siennes).

Étape 13 – Tests, validation, CI

Objectif :
- Vérifier la cohérence générale avant déploiement.

Attendu :
- Tests automatisés minimum (ou script manuel documenté) pour :
• création utilisateur → profil
• création locataire + logement
• création ticket
• diffusion ticket
• acceptation entreprise
• création mission
• assignation technicien
• clôture mission
• facturation
• notifications.

L ' IA doit proposer un plan de test et, idéalement, du code de test.

Vérifications :
- Tous les scénarios métiers principaux passent sans erreur.
- Les accès sont cohérents côté RLS.
- Les erreurs sont gérées proprement.

7. COMPORTEMENT EN CAS D ' ERREUR

À n ' importe quelle étape :

1. Si une erreur technique est envisagée (conflit RLS, incohérence, doublon, risque de fuite de données) :

- l ' IA stoppe immédiatement le déroulement.
- l ' IA décrit précisément :

• la cause probable,

• l ' impact,

• la ou les solutions possibles.

2. L ' IA propose une version corrigée du code ou du schéma, mais n ' avance pas d ' étape.
3. L ' IA attend la validation explicite de l ' utilisateur avant de passer à l ' étape suivante.

8. REPRISE APRÈS CORRECTION

Quand l ' utilisateur indique qu ' il a corrigé un problème (par exemple en disant " correction faite, continue étape X ") :

1. L ' IA :

- réévalue rapidement le contexte
- rappelle brièvement ce qu ' elle était en train de faire
- reprend à l ' étape en question, sans sauter ce qui n ' a pas été validé.

2. L ' IA ne recommence pas le projet depuis zéro sauf demande explicite.

FIN DU DOCUMENT.