

JETC_IMMO

DOCUMENT MAÎTRE D'EXÉCUTION TECHNIQUE

Mode DEMO / Mode PRO / Accès ADMIN JTEC

Source de vérité unique – Version exécutable

1. RÔLE ET COMPORTEMENT DE L'IA

L'IA agissant sur le projet **JETC_IMMO** est une IA de type développeur (Copilot, Codex ou équivalent).

Elle n'est **pas autorisée** à :

- improviser
- optimiser sans instruction
- proposer des alternatives non demandées
- sauter des étapes
- “faire au mieux”

 **Elle exécute ce document strictement, dans l'ordre, sans interprétation.**

1.1 RÈGLES DE COMPORTEMENT OBLIGATOIRES

1. L'IA **doit suivre les étapes dans l'ordre**, de l'Étape 0 à l'étape finale.
2. L'IA **ne passe JAMAIS** à l'étape suivante tant que :
 - le code de l'étape courante n'est pas cohérent
 - les vérifications définies ne sont pas validées
3. À **chaque étape**, l'IA doit :
 - expliquer ce qu'elle fait
 - fournir le **code complet** (backend, SQL, frontend)
 - indiquer **où placer chaque fichier**
 - expliquer **comment tester** (local + DEMO si applicable)
4. En cas :
 - d'ambiguïté
 - d'incohérence

- de conflit DEMO / PRO
 - de doute RLS / sécurité
- 👉 **l'IA s'arrête immédiatement**

5. L'IA écrit un code :

- lisible
- commenté
- structuré
- avec séparation claire des responsabilités

6. La base de données est **obligatoirement Supabase** :

- PostgreSQL
- Auth
- RLS
- Storage

7. Les variables d'environnement sont **déclarées explicitement dès leur apparition**

8. La distinction **MODE DEMO / MODE PRO** est respectée pour :

- données
- routes API
- paramètres
- comportements sensibles (paiement, email, notifications)

👉 **Toute violation de ces règles est considérée comme un BUG.**

1.2 PRINCIPE FONDAMENTAL DE CONCEPTION

Le système doit être **pensé entièrement dès le départ**.

- ✗ Pas de “on verra plus tard”
- ✗ Pas de table provisoire
- ✗ Pas de RLS ajoutée en fin de projet
- ✗ Pas de DEMO bricolé

👉 Une fois une étape validée, elle est considérée comme finale.

1.3 OBJECTIF GLOBAL

Construire une application **JETC_IMMO** permettant de gérer :

- Régies
- Propriétaires
- Immeubles
- Logements
- Locataires
- Entreprises
- Techniciens
- Tickets d'intervention
- Missions
- Facturation
- Messagerie
- Notifications
- Abonnements / modules payants
- Vue administrateur JTEC

Avec :

- un **niveau de sécurité RLS strict**
 - un **MODE DEMO totalement isolé**
 - une architecture **pérenne et extensible**
-

👉 STOP ICI

PARTIE II – CONTEXTE MÉTIER & MODÈLE FONCTIONNEL

(Version réécrite, exhaustive, définitive)

2. CONTEXTE MÉTIER GÉNÉRAL

JETC_IMMO est une application de **gestion complète des interventions techniques immobilières**, servant d'intermédiaire structuré entre :

- les **locataires** (demandeurs),
- les **régies** (décideurs),
- les **entreprises** (prestataires),
- les **techniciens** (exécutants),
- et **JETC** (opérateur de la plateforme).

Le système couvre **l'intégralité du cycle de vie** d'un problème technique, depuis sa déclaration jusqu'à la facturation finale, en conservant **l'historique, les échanges et les preuves**.

2.1 ACTEURS MÉTIER

2.1.1 Locataire

Le locataire :

- est rattaché à un **logement**
- peut déclarer des **tickets d'intervention**
- peut suivre l'état de ses tickets
- peut échanger via la messagerie
- peut signer une intervention terminée

Limitations :

- ne voit **que ses propres données**
 - ne choisit **pas l'entreprise**
 - ne voit **pas les données financières**
-

2.1.2 Régie

La régie :

- gère un **portefeuille d'immeubles**
- gère les **logements** associés
- associe des **locataires** aux logements

- reçoit et **valide les tickets**
- décide de la **diffusion des tickets**
- consulte les missions et factures

Pouvoirs clés :

- choix du **mode de diffusion** des tickets
- accès aux données agrégées de son périmètre

Limitations :

- ne voit **que ses propres immeubles**
 - n'accède jamais aux données d'autres régies
-

2.1.3 Entreprise

L'entreprise :

- reçoit des tickets selon les règles de diffusion
- peut **accepter ou refuser** une intervention
- crée une **mission** en acceptant un ticket
- gère ses **techniciens**
- consulte ses missions et factures

Limitations :

- ne voit que les tickets **auxquels elle est autorisée**
 - ne peut modifier que **ses propres missions**
-

2.1.4 Technicien

Le technicien :

- est rattaché à **une seule entreprise**
- voit uniquement les **missions qui lui sont assignées**
- peut :
 - démarrer une mission
 - signaler un retard

- ajouter des photos
- rédiger un rapport
- faire signer le locataire

Limitations :

- aucun accès aux tickets non assignés
 - aucun accès aux données financières globales
-

2.1.5 Propriétaire

Le propriétaire :

- est lié à un ou plusieurs immeubles
- a un rôle **consultatif**
- accède uniquement à :
 - des rapports
 - des historiques
 - des données non opérationnelles

Limitations :

- ne crée jamais de ticket
 - ne pilote jamais d'intervention
-

2.1.6 Administrateur JTEC

L'administrateur JTEC :

- possède le rôle **admin_jtec**
- a une **vue globale** de la plateforme
- accède uniquement à :
 - des **statistiques agrégées**
 - des indicateurs business
 - des données anonymisées

Interdictions absolues :

- accès direct aux données nominatives sans besoin métier explicite
 - contournement des RLS
-

2.2 OBJETS MÉTIER FONDAMENTAUX

2.2.1 Immeuble

Un immeuble :

- appartient à **une régie**
 - contient **un ou plusieurs logements**
 - peut être lié à **un ou plusieurs propriétaires**
-

2.2.2 Logement

Un logement :

- appartient à **un immeuble**
 - est rattaché à **une régie**
 - peut avoir :
 - plusieurs locataires dans le temps
 - un seul locataire actif à un instant donné
-

2.2.3 Ticket

Un ticket représente :

- une **demande d'intervention**
- créée par un locataire
- toujours liée à :
 - un logement
 - une régie

Un ticket a un **cycle de vie** clair :

- nouveau
- validé

- diffusé
 - accepté
 - clos
-

2.2.4 Mission

Une mission :

- est créée lorsqu'une entreprise accepte un ticket
- est rattachée :
 - à un ticket
 - à une entreprise
 - à un technicien (optionnel au départ)

Une mission évolue :

- en attente
 - planifiée
 - en cours
 - terminée
-

2.2.5 Facture

Une facture :

- est générée à partir d'une mission terminée
 - appartient à une entreprise
 - est consultable par la régie
 - inclut une **commission JETC**
-

2.3 MODES DE FONCTIONNEMENT : DEMO / PRO

2.3.1 MODE DEMO (CRITIQUE)

Le MODE DEMO est :

- accessible uniquement aux **visiteurs non connectés**

- totalement isolé
- sans aucune écriture en base
- sans appel API réel

Caractéristiques :

- données fictives
- actions simulées
- stockage local (localStorage)
- aucune pollution PRO possible

👉 Le MODE DEMO existe pour **démontrer**, pas pour tricher.

2.3.2 MODE PRO

Le MODE PRO :

- utilise de **vraies données**
- s'appuie sur Supabase Auth
- applique strictement les RLS
- déclenche les vraies règles métier

Toute action en PRO est :

- persistée
- historisée
- traçable

2.4 RÈGLES MÉTIER TRANSVERSES

Règles non négociables :

- Un ticket appartient toujours à **une seule régie**
- Une mission appartient toujours à **une seule entreprise**
- Un technicien n'intervient que pour **son entreprise**
- Un utilisateur ne voit jamais ce qui ne lui appartient pas
- DEMO et PRO **ne se croisent jamais**

2.5 OBJECTIF DE CONCEPTION

Le modèle métier doit permettre :

- une **extension future** (paiement, SLA, contrats)
 - une **séparation claire des responsabilités**
 - une **lecture simple pour l'IA**
 - une **sécurité native**, pas ajoutée après
-

👉 FIN DE LA PARTIE II

PARTIE III – ARCHITECTURE TECHNIQUE GLOBALE

(Version réécrite, structurée, définitive)

3. OBJECTIF DE L'ARCHITECTURE

L'architecture de **JETC_IMMO** doit garantir simultanément :

- **Sécurité native** (RLS Supabase comme dernier rempart)
- **Séparation stricte** frontend / backend / base
- **Isolation totale DEMO / PRO**
- **Lisibilité pour une IA exécutante**
- **Évolutivité** sans refonte complète

Aucune décision d'architecture ne doit :

- court-circuiter la sécurité
 - mélanger les responsabilités
 - dépendre d'un “on verra plus tard”
-

3.1 STACK TECHNIQUE IMPOSÉE

3.1.1 Backend

- **Supabase**
 - PostgreSQL

- Auth
- Row Level Security (RLS)
- Storage
- **API Routes**
 - Hébergées sur **Vercel**
 - Préfixe obligatoire /api/*

👉 Le backend est le **seul** autorisé à effectuer :

- des opérations sensibles
- des écritures complexes
- des accès via service_role

3.1.2 Frontend

- Application web :
 - HTML / JS
 - ou Next.js (migration possible, non obligatoire au départ)
- Rôles :
 - affichage
 - interaction utilisateur
 - appels API sécurisés

Interdictions absolues côté frontend :

- utilisation de la clé service_role
- contournement des RLS
- accès direct aux buckets privés

3.1.3 Authentification

- **Supabase Auth**
 - email / password
 - Magic Link réservé à l'ADMIN JTEC

- Les rôles sont gérés via la table profiles
 - Le frontend **ne décide jamais** du rôle effectif
-

3.1.4 Stockage de fichiers

- **Supabase Storage**
 - Buckets privés uniquement
 - Accès :
 - lecture / écriture via API backend
 - jamais directement depuis le frontend
-

3.2 PRINCIPES ARCHITECTURAUX NON NÉGOCIABLES

3.2.1 Principe de défiance totale

- Le frontend est considéré comme **non fiable**
- Le backend vérifie **toutes** les intentions
- La base bloque **tout ce qui passe**

👉 Si une règle n'est pas implémentée en RLS, elle est considérée comme inexisteante.

3.2.2 Séparation des responsabilités

Couche Responsabilités

Frontend UI, navigation, UX, appels API

Backend API Vérifications métier, sécurité, orchestration

Supabase Données, contraintes, RLS, intégrité

Storage Fichiers, preuves, signatures

Aucune couche ne doit faire le travail d'une autre.

3.2.3 Architecture DEMO / PRO

Le projet doit clairement distinguer :

- **MODE DEMO**
 - aucune base réelle
 - aucune API réelle
 - aucune persistance serveur
- **MODE PRO**
 - données persistées
 - règles métier actives
 - RLS appliquées

👉 Toute ambiguïté DEMO / PRO est un **BUG CRITIQUE**.

3.3 ARBORESCENCE DE PROJET (RÉFÉRENCE)

Arborescence cible minimale :

```
/src
  /components
  /pages
  /lib
  /context

  /api
    /auth
    /admin
    /files
    healthcheck.js

  /supabase
    /schema
    /policies
    /demo
```

/public

Chaque dossier a un rôle précis.

Aucun fichier “fourre-tout”.

3.4 GESTION DES VARIABLES D'ENVIRONNEMENT

3.4.1 Fichiers requis

- .env.example → **commit**
 - .env.local → **jamais commit**
-

3.4.2 Variables obligatoires

SUPABASE_URL=

SUPABASE_ANON_KEY=

SUPABASE_SERVICE_ROLE_KEY=

MODE=demo

3.4.3 Règles d'utilisation

- SUPABASE_ANON_KEY
 - frontend uniquement
 - SUPABASE_SERVICE_ROLE_KEY
 - backend API uniquement
 - MODE
 - conditionne DEMO / PRO
 - utilisé dans API + SQL (RLS)
-

3.5 CLIENTS SUPABASE

3.5.1 Client frontend

- Fichier : /src/lib/supabaseClient.js

- Utilise :
 - SUPABASE_URL
 - SUPABASE_ANON_KEY

Règle :

Ce client **ne doit jamais** être utilisé côté serveur.

3.5.2 Client backend

- Fichier : /api/_supabase.js
- Utilise :
 - SUPABASE_SERVICE_ROLE_KEY

Règle :

Ce client **ne doit jamais** être exposé au frontend.

3.6 ROUTER API – PRINCIPES

3.6.1 Règles générales

- Toutes les routes sensibles passent par /api/*
- Chaque route :
 - vérifie l'authentification
 - vérifie le rôle
 - applique les règles métier

3.6.2 Route de santé (obligatoire)

- /api/healthcheck
- Retour :

```
{ "ok": true }
```

Utilité :

- test de déploiement
- test local

- test DEMO
-

3.7 LOGS & OBSERVABILITÉ

Principes :

- logs explicites
- logs lisibles
- préfixes clairs ([AUTH], [ADMIN], [DEMO], [ERROR])

Le logging n'est **pas optionnel**, c'est un outil de survie.

3.8 OBJECTIF FINAL DE L'ARCHITECTURE

À la fin :

- aucune couche n'a de pouvoir excessif
- aucun secret n'est exposé
- aucune règle critique n'est uniquement côté frontend
- le système peut évoluer sans refonte

👉 L'architecture doit **tenir dans le temps**, pas juste passer la démo.

👉 **FIN DE LA PARTIE III**

PARTIE IV – MODE DEMO

(*Spécification complète, isolation totale, exécution stricte*)

4.1 OBJECTIF DU MODE DEMO

Le MODE DEMO permet à un visiteur **non connecté** d'explorer JETC_IMMO :

- sans inscription
- sans connexion
- sans données réelles
- sans accès base de données

- sans déclenchement d'actions sensibles

Le MODE DEMO doit permettre une démonstration “complète” des vues :

- Locataire
- Régie
- Entreprise
- Technicien
- Admin JTEC (version démo “statique”)

👉 La démo est une **simulation UI**, pas une plateforme parallèle.

4.2 RÈGLE MÉTIER CENTRALE (NON NÉGOCIABLE)

Le MODE DEMO est accessible **UNIQUEMENT** aux visiteurs **non connectés**.

- user == null → DEMO autorisé ✅
- user != null → DEMO interdit ❌ (même si l'utilisateur est “non abonné”)

Conséquence :

- si un utilisateur est connecté, il n'a **jamais** le droit d'accéder au DEMO.
 - s'il essaye d'accéder à /demo/*, il est redirigé automatiquement vers le flux PRO (dashboard ou pricing selon logique).
-

4.3 INTERDICTIONS ABSOLUES EN DEMO

En MODE DEMO, il est interdit de :

- appeler Supabase Database (select/insert/update/delete)
- appeler Supabase Storage
- appeler des routes /api/* qui écrivent en base
- déclencher email réel (Auth)
- déclencher paiement réel
- déclencher notification réelle (mail/sms/push)

👉 Toute action UI en DEMO est **simulée**.

4.4 STOCKAGE DEMO (LOCAL UNIQUEMENT)

Le MODE DEMO utilise **uniquement le navigateur** :

- localStorage
- éventuellement sessionStorage (optionnel)

Aucune donnée DEMO n'est envoyée au serveur.

4.4.1 Clés localStorage obligatoires

- jetc_demo_mode : "true" | "false"
- jetc_demo_role : "regie" | "entreprise" | "locataire" | "technicien" | "admin_jtec_demo"
- jetc_demo_profile : JSON stringifié du profil démo
- jetc_demo_session : JSON stringifié d'une session fictive

Règle :

- ces clés doivent être **nettoyées complètement** à la sortie DEMO.
-

4.5 PROFILS DEMO (PRÉDÉFINIS ET STABLES)

Les profils DEMO sont statiques (définis dans le code).

Ils doivent inclure au minimum :

- id (string stable type "DEMO_USER_001")
- email (fictif)
- role
- is_demo: true
- références “fictives” :
 - regie_id, entreprise_id, logement_id, etc.

4.5.1 Exemple (structure obligatoire)

```
const roleProfiles = {  
    regie: {  
        id: "DEMO_USER_001",  
        email: "demo.regie@jetc-immo.local",  
        role: "regie",  
    },  
}
```

```
    regie_id: "REGIE_DEMO_001",
    is_demo: true,
    permissions: ["demo_full_access"]
},
entreprise: {
    id: "DEMO_USER_002",
    email: "demo.entreprise@jetc-immo.local",
    role: "entreprise",
    entreprise_id: "ENTREPRISE_DEMO_001",
    is_demo: true,
    permissions: ["demo_full_access"]
},
locataire: {
    id: "DEMO_USER_003",
    email: "demo.locataire@jetc-immo.local",
    role: "locataire",
    logement_id: "LOGEMENT_DEMO_001",
    is_demo: true,
    permissions: ["demo_full_access"]
},
technicien: {
    id: "DEMO_USER_004",
    email: "demo.technicien@jetc-immo.local",
    role: "technicien",
    entreprise_id: "ENTREPRISE_DEMO_001",
    is_demo: true,
    permissions: ["demo_full_access"]
},
```

```
admin_jtec_demo: {  
    id: "DEMO_ADMIN_001",  
    email: "demo.admin@jetc-immo.local",  
    role: "admin_jtec_demo",  
    is_demo: true,  
    permissions: ["demo_full_access"]  
}  
};
```

4.6 ACTIVATION DU MODE DEMO (FLUX)

4.6.1 Point d'entrée DEMO

Sur la landing page / :

- bouton “**Découvrir en mode DEMO**”
- ouverture d'un **hub DEMO** (sélection de rôle)

4.6.2 Hub DEMO

Page : /demo/demo-hub

- choix du rôle
- description rapide
- bouton “entrer en démo”

4.6.3 Activation (fonction obligatoire)

But : remplir localStorage + définir un état DEMO global.

Pseudo-contrat :

1. set jetc_demo_mode = true
 2. set jetc_demo_role = role
 3. set jetc_demo_profile = profile(role)
 4. set jetc_demo_session = session_fictive
 5. rediriger vers le dashboard du rôle
-

4.7 PROTECTION DES PAGES DEMO (OBLIGATOIRE)

Toutes les pages DEMO sont protégées par un guard unique :

- HOC withDemoAccess
- ou middleware/guard global (selon stack)

4.7.1 Règle de withDemoAccess

- si utilisateur PRO connecté → redirection immédiate (DEMO interdit)
- si DEMO non activé → redirection /demo/demo-hub
- si DEMO activé → autorisation

Ce guard s'exécute :

- à chaque navigation
 - à chaque refresh
-

4.8 STRUCTURE DES PAGES DEMO (OBLIGATOIRE)

Chemins imposés :

/pages/demo

demo-hub.js

/regie

dashboard.js

immeubles.js

logements.js

tickets.js

interventions.js

parametres.js

/entreprise

dashboard.js

missions.js

techniciens.js

planning.js

statistiques.js

/locataire

dashboard.js

tickets.js

nouveau-ticket.js

profil.js

/technicien

dashboard.js

missions.js

planning.js

historique.js

profil.js

/admin

dashboard.js (démo: stats simulées)

4.9 DONNÉES SIMULÉES (ZÉRO API)

Les données DEMO sont :

- des constantes JS
- ou des fichiers JSON
- jamais une query Supabase

4.9.1 Règles pour les actions simulées

Exemples :

- “Créer un ticket” en DEMO :

- affiche une confirmation
- ajoute dans une liste locale optionnelle (non persistée serveur)
- redirige vers la liste

Chaque action doit afficher :

- “Ceci est une simulation”
 - “Aucune donnée réelle n'a été enregistrée”
-

4.10 UI DEMO (OBLIGATOIRE)

4.10.1 Bandeau DEMO global

Visible sur toutes les pages DEMO.

Contient :

- badge “🎭 MODE DEMO”
- rôle courant
- bouton “Changer de rôle”
- bouton “Quitter la démo”

Position :

- fixed en haut
 - z-index élevé
-

4.11 SORTIE DU MODE DEMO (NETTOYAGE TOTAL)

4.11.1 Quitter la démo

Action :

- supprimer toutes les clés DEMO
- reset état DEMO
- rediriger vers /

4.11.2 Nettoyage complet (fonction obligatoire)

La fonction doit :

- supprimer les clés connues

- supprimer toute clé contenant demo (fallback)
- loguer la purge

Aucune trace DEMO ne doit survivre.

4.12 TRANSITION DEMO → PRO

Lorsqu'un utilisateur clique "S'inscrire" ou "Se connecter" depuis DEMO :

- nettoyage DEMO immédiat
 - redirection vers /login ou /register
 - aucune donnée DEMO transférée
-

4.13 VALIDATION & TESTS DEMO (CHECKLIST)

Tests obligatoires :

1. Visiteur non connecté → peut entrer en DEMO
 2. Switch de rôle → fonctionne
 3. Refresh page DEMO → état conservé (localStorage)
 4. Quitter DEMO → purge totale
 5. Utilisateur connecté → accès DEMO interdit (redirection)
 6. Aucun appel Supabase / API write en DEMO (vérifier logs réseau)
-

FIN DE LA PARTIE IV

5.6 API PRO : ROUTES AUTH OBLIGATOIRES

Toutes les routes Auth sont sous :

- /api/auth/*

5.6.1 POST /api/auth/register

But :

- créer utilisateur
- vérifier création profil (ou le créer si fallback)
- retourner un résultat clair

Entrées :

- email
- password
- (optionnel) type d'inscription (si flux régie/entreprise plus tard)

Sorties :

- success true/false
- userId
- profil minimal (role, ids)

Erreurs obligatoires :

- email invalide
 - mot de passe invalide
 - utilisateur existe déjà
 - erreur Supabase Auth
-

5.6.2 POST /api/auth/login

But :

- authentifier
- charger le profil
- retourner user + profil

Entrées :

- email
- password

Sorties :

- userId
- role
- regie_id / entreprise_id
- token/session (selon méthode retenue)

Erreurs obligatoires :

- mauvais mot de passe
 - utilisateur inconnu
 - profil absent
 - accès refusé (cas admin, etc.)
-

5.6.3 GET /api/auth/me

But :

- renvoyer le profil courant (session obligatoire)

Sorties :

- profil complet (ou minimal selon besoin)
- role
- ids

Erreurs :

- pas de session
 - session invalide/expirée
 - profil absent
-

5.7 GESTION DES SESSIONS (PRO)

5.7.1 Source de session

- session Supabase (JWT)
- stockée côté client (selon SDK)
- jamais un “role” stocké et cru aveuglément

5.7.2 Règle de sécurité

Si localStorage contient un rôle falsifié :

- le backend doit refuser
 - les RLS doivent bloquer
-

5.8 REDIRECTIONS PRO (ROUTING)

Après login réussi :

- redirection vers dashboard selon rôle :

role route

locataire /locataire/dashboard

regie /regie/dashboard

entreprise /entreprise/dashboard

technicien /technicien/dashboard

proprietaire /proprietaire/dashboard

admin_jtec /admin

Si rôle inconnu :

- redirection / + message erreur
-

5.9 INSCRIPTION SPÉCIFIQUE (OPTION CADRE)

Le document prévoit qu'une régie/entreprise puisse avoir un flux spécifique.

Règle :

- pas d'assignation “magique” du rôle côté frontend
- si un flux spécifique existe :
 - backend valide la demande
 - profil mis à jour côté serveur
 - RLS toujours appliquée

👉 À ce stade (fondations), l'inscription par défaut reste locataire.

5.10 MODÈLE “ABONNEMENTS / MODULES” (LIEN PRO)

Le PRO doit anticiper les modules payants :

- un utilisateur PRO peut être :
 - inscrit mais non abonné (accès limité)
 - abonné (accès modules)

Règle :

- la logique d'accès modules doit être vérifiée :
 - backend
 - et/ou vues SQL
 - et RLS si nécessaire

(Les tables plans & subscriptions arrivent à l'Étape 13.)

5.11 TESTS DE VALIDATION (PRO)

Checklist minimale :

1. register → utilisateur créé
2. trigger → profil créé automatiquement
3. login → session + profil chargés
4. /api/auth/me → renvoie profil
5. redirection → dashboard correct
6. accès à une donnée d'une autre régie → bloqué (RLS)
7. accès DEMO une fois connecté → interdit + purge

PARTIE V – MODE PRO

- 5.1 Objectif
- 5.2 Règles PRO
- 5.3 Auth
- 5.4 Profiles
- 5.5 Création automatique
- 5.6 API Auth
- 5.7 Sessions
- 5.8 Routing
- 5.9 Inscriptions spécifiques
- **5.10 Abonnements, packs et limites (CE QUE TU VIENS D'AJOUTER)**
- 5.11 Multilingue (login, landing, persistance langue)

- 5.12 Tests PRO

1. PRINCIPE GLOBAL (NON NÉGOCIABLE)

👉 Le pack choisi détermine automatiquement le formulaire.

L'utilisateur ne choisit pas tout, il confirme ce qui est autorisé.

- Essentiel → contraintes strictes
- Pro → contraintes élargies
- Premium → pas de contraintes

Aucun champ inutile.

Aucune promesse non tenue.

Aucune surprise après paiement.

2. PACKS & RÈGLES MÉTIER (SOURCE DE VÉRITÉ)

2.1 Packs disponibles

Pack	Prix	Logements	Utilisateurs	Entreprises
Essentiel	49 €	25 max	1	5 max
Pro	99 €	150 max	5 max	illimité
Premium	199 €	illimité	illimité	illimité

👉 Ces règles doivent exister :

- dans le frontend (UX)
 - dans le backend (API)
 - dans la base (contrôles + RLS si nécessaire)
-

3. PARCOURS UTILISATEUR (INSCRIPTION)

3.1 Page d'accueil (multilingue)

Landing page :

- Sélecteur de langue visible **avant toute action**
- Langues disponibles :
 - FR Français (par défaut si navigateur FR)

- GB English

- DE Deutsch

Le choix de langue :

- est stocké (cookie ou localStorage)
- est utilisé immédiatement
- est repris automatiquement au login
- reste modifiable à tout moment

👉 La langue **ne dépend pas du compte**, mais du choix utilisateur.

3.2 Sélection du pack

Sur la landing :

- cartes Essentiel / Pro / Premium
- clic sur un pack = redirection vers /register?plan=essentiel|pro|premium

Le pack est :

- **pré-sélectionné**
 - **non modifiable** dans le formulaire (pour éviter le bricolage)
 - modifiable uniquement en revenant à la page pricing
-

4. FORMULAIRE D'INSCRIPTION (DYNAMIQUE PAR PACK)

4.1 Champs communs (tous les packs)

- Nom de la régie
 - Email administrateur
 - Mot de passe
 - Téléphone
 - Pays
 - Langue préférée (pré-remplie selon choix initial)
-

4.2 Champs conditionnés par le pack

Pack ESSENTIEL (25 logements)

- Nombre de logements
 - champ numérique
 - max = 25
 - message clair :

“Ce pack est limité à 25 logements maximum.”

- Nombre d'utilisateurs
 - champ désactivé
 - valeur forcée = 1
- Nombre d'entreprises partenaires
 - champ numérique
 - max = 5

Tout dépassement :

- bloqué côté UI
 - bloqué côté API
 - message explicite (pas un “error 400” débile)
-

Pack PRO (150 logements)

- Nombre de logements
 - champ numérique
 - max = 150
 - Nombre d'utilisateurs
 - champ numérique
 - max = 5
 - Entreprises partenaires
 - pas de limite affichée
 - info textuelle : “illimité”
-

Pack PREMIUM (illimité)

- Aucun champ de limite
- Affichage :

“Aucune limite de logements, d'utilisateurs ou d'entreprises.”

Le formulaire est volontairement plus simple.

Premium = confort, pas paperasse.

5. VALIDATION & CRÉATION COMPTE

À la soumission :

1. Vérification frontend (UX)
2. Vérification backend (API)
3. Création :
 - utilisateur admin
 - régie
 - abonnement associé
4. Stockage :
 - plan
 - max_logements
 - max_users
 - max_partners
 - language

👉 Les limites **ne sont jamais recalculées à la volée.**

Elles sont stockées clairement.

6. MULTILINGUE – RÈGLES STRICTES

6.1 Langue au login

- La langue choisie sur la landing :
 - est mémorisée
 - est appliquée au login
 - est appliquée après connexion

Si utilisateur change la langue après login :

- mise à jour du profil
- persistance immédiate

6.2 Traductions obligatoires

Tout texte visible doit exister en :

- FR
- EN
- DE

Y compris :

- noms de packs
- descriptions
- messages d'erreur
- limites (“jusqu'à 25 logements”)

👉 Aucune chaîne en dur dans le code.

7. ANTICIPATION ÉVOLUTIVE (IMPORTANT)

Ce système permet plus tard :

- ajout de packs intermédiaires
- upgrade / downgrade
- facturation proratisée
- modules optionnels

Sans :

- casser les comptes existants
- réécrire les formulaires
- bidouiller les limites à la main

8. RÈGLE D'OR (À GRAVER)

Le pack décide.

Le formulaire s'adapte.

L'utilisateur confirme.

Le système applique.

La base verrouille.

PARTIE VI – SUPABASE

6. STRUCTURE DE DONNÉES & SCHÉMA GLOBAL

6.1 PRINCIPES FONDAMENTAUX (À RESPECTER)

1. **Une table = une responsabilité**
 2. Pas de données calculées stockées inutilement
 3. Toutes les relations sont explicites (FK)
 4. Tout ce qui est sensible sera protégé par RLS (étape 7)
 5. Les packs, limites et langues sont **stockés**, jamais “déduits”
-

6.2 ORGANISATION DES FICHIERS SQL

Structure imposée du dossier /supabase :

/supabase

/schema

01_extensions.sql

02Enums.sql

03_core.sql

04_users.sql

05_immo.sql

06_tickets.sql

07_missions.sql

08_billing.sql

09_demo.sql

/policies

(étape 7)

Chaque fichier :

- peut être exécuté indépendamment
 - ne contient QUE son périmètre
 - aucun “fourre-tout”
-

6.3 EXTENSIONS & BASE (01_extensions.sql)

```
create extension if not exists "uuid-ossp";
```

```
create extension if not exists "pgcrypto";
```

6.4 ENUMS (02_enums.sql)

```
create type user_role as enum (
```

```
    'locataire',
```

```
    'regie',
```

```
    'entreprise',
```

```
    'technicien',
```

```
    'proprietaire',
```

```
    'admin_jtec'
```

```
);
```

```
create type plan_type as enum (
```

```
    'essentiel',
```

```
    'pro',
```

```
    'premium'
```

```
);
```

```
create type ticket_status as enum (
```

```
    'nouveau',
```

```
    'valide',
```

```
'diffuse',
'accepte',
'clos'
);

create type mission_status as enum (
'en_attente',
'planifiee',
'en_cours',
'terminee'
);
```

6.5 TABLE profiles (04_users.sql)

```
create table profiles (
id uuid primary key references auth.users(id) on delete cascade,
email text not null,
role user_role not null default 'locataire',
language text not null default 'fr',
is_demo boolean not null default false,
regie_id uuid,
entreprise_id uuid,
created_at timestamptz default now()
);
```

- 👉 La langue est stockée ici
- 👉 Le rôle aussi
- 👉 Aucune logique frontend ne remplace ça

6.6 TABLE regies (05_immo.sql)

```
create table regies (
    id uuid primary key default uuid_generate_v4(),
    name text not null,
    plan plan_type not null,
    max_logements integer,
    max_users integer,
    max_partners integer,
    created_at timestamptz default now()
);
```

Règle :

- max_* = NULL → illimité (Premium)
 - Ces valeurs viennent **du formulaire PRO**
-

6.7 TABLES IMMO DE BASE

```
create table immeubles (
    id uuid primary key default uuid_generate_v4(),
    regie_id uuid not null references regies(id),
    name text not null
);

create table logements (
    id uuid primary key default uuid_generate_v4(),
    immeuble_id uuid not null references immeubles(id),
    reference text
```

```
);
```

6.8 TABLE tickets (06_tickets.sql)

```
create table tickets (
    id uuid primary key default uuid_generate_v4(),
    logement_id uuid not null references logements(id),
    regie_id uuid not null references regies(id),
    created_by uuid not null references profiles(id),
    status ticket_status not null default 'nouveau',
    description text,
    created_at timestamptz default now()
);
```

6.9 TABLE missions (07_missions.sql)

```
create table missions (
    id uuid primary key default uuid_generate_v4(),
    ticket_id uuid not null references tickets(id),
    entreprise_id uuid not null,
    technicien_id uuid,
    status mission_status not null default 'en_attente',
    report text,
    created_at timestamptz default now()
);
```

6.10 TABLE BILLING (08_billing.sql – base)

```
create table subscriptions (
    id uuid primary key default uuid_generate_v4(),
    regie_id uuid not null references regies(id),
    plan plan_type not null,
    active boolean not null default true,
    started_at timestamptz default now()
);
```

(Paiement détaillé viendra plus tard, structure OK.)

6.11 TABLES DEMO (09_demo.sql)

Optionnel, mais **aucune donnée PRO** dedans.
Ou totalement vide si DEMO = 100% frontend.

👉 FIN DE L'ÉTAPE 6

PARTIE VII – RLS TABLE PAR TABLE

(Sécurité des données – SQL uniquement – MODE DEMO / MODE PRO respecté)

7.1 RÔLE DE CETTE PARTIE

Cette partie définit **exclusivement** les règles de **Row Level Security (RLS)** appliquées à la base de données Supabase.

👉 Elle ne décrit :

- ni les routes API,
- ni la logique frontend,
- ni les flux métiers applicatifs.

Son unique objectif est de garantir que **même en cas de fuite de clé publique, aucun utilisateur ne puisse accéder à des données qui ne lui appartiennent pas.**

La RLS est le **dernier rempart**, non négociable.

7.2 PRINCIPES GÉNÉRAUX RLS (OBLIGATOIRES)

Les règles suivantes s'appliquent à **toutes les tables concernées** :

- 1. Row Level Security activée sur toutes les tables**
- 2. Aucun accès anonyme** aux données sensibles
- 3. Chaque policy doit être :**
 - explicite,
 - lisible,
 - commentée,
 - limitée à un rôle précis
- 4. Les règles doivent respecter strictement** le modèle métier défini en PARTIE II
- 5. Le MODE DEMO ne doit jamais permettre :**
 - la lecture de données PRO,
 - l'écriture de données PRO,
 - la contamination croisée DEMO / PRO

7.3 ORGANISATION DES FICHIERS RLS

Les règles RLS sont organisées **table par table**, dans des fichiers SQL distincts.

Arborescence imposée :

/supabase/policies/
10_policies_profiles.sql
11_policies_regies.sql
12_policies_locataires.sql
13_policies_entreprises.sql
14_policies_techniciens.sql
15_policies_tickets.sql
16_policies_missions.sql
17_policies_factures.sql
18_policies_messages.sql

👉 Une table = un fichier

👉 Aucun regroupement autorisé

7.4 GESTION DES RÔLES (RÉFÉRENCE)

Les rôles applicatifs utilisés par la RLS sont :

- locataire
- regie
- entreprise
- technicien
- propriétaire (lecture uniquement)
- admin_jtec

Ces rôles sont stockés dans la table profiles et **ne sont jamais déduits côté frontend.**

7.5 TABLE profiles – RÈGLES RLS

Objectif :

- un utilisateur ne peut accéder **qu'à son propre profil**

Règles :

- un utilisateur connecté peut :
 - lire son profil
 - mettre à jour ses paramètres non sensibles (ex : langue, préférences)
- un utilisateur ne peut jamais :
 - modifier son rôle
 - modifier son rattachement (regie_id, entreprise_id)
- aucune lecture globale des profils n'est autorisée

Le rôle admin_jtec **n'a pas d'accès direct** aux profils nominaux par défaut.

7.6 TABLES STRUCTURELLES (RÉGIES / IMMEUBLES / LOGEMENTS / LOCATAIRES)

7.6.1 Règles communes

- **Une régie :**
 - ne voit que les données rattachées à **sa propre régie**
 - **Un locataire :**
 - ne voit que son logement et ses données personnelles
 - **Une entreprise :**
 - n'a aucun accès direct aux structures immobilières
 - **Un technicien :**
 - n'a aucun accès aux structures immobilières
-

7.6.2 Propriétaire

Le rôle propriétaire :

- dispose d'un accès **consultatif uniquement**
 - ne peut modifier aucune donnée
 - ne voit que les immeubles auxquels il est explicitement lié
-

7.7 TABLE tickets – RÈGLES RLS

Accès en lecture

- **Locataire :**
 - voit uniquement les tickets liés à son logement
- **Régie :**
 - voit uniquement les tickets de sa régie
- **Entreprise :**
 - voit les tickets accessibles selon :
 - le mode de diffusion (général ou restreint)
 - les règles définies à l'ÉTAPE 6
- **Technicien :**
 - n'a pas accès aux tickets non transformés en mission
- **Admin JTEC :**

- pas d'accès direct aux tickets nominaux
-

Accès en écriture

- **Création :**
 - autorisée uniquement pour un locataire
 - uniquement pour son logement
- **Modification :**
 - autorisée uniquement si cohérente avec le rôle :
 - régie pour validation/diffusion
 - entreprise uniquement si une mission existe

7.8 TABLE missions – RÈGLES RLS

Accès en lecture

- **Entreprise :**
 - voit uniquement ses missions
- **Technicien :**
 - voit uniquement les missions qui lui sont assignées
- **Régie :**
 - voit les missions liées aux tickets de sa régie
- **Locataire :**
 - voit uniquement la mission liée à son ticket
- **Admin JTEC :**
 - accès indirect uniquement via vues agrégées

Accès en écriture

- **Création :**
 - autorisée uniquement pour une entreprise
 - uniquement si elle est autorisée à accepter le ticket

- **Modification :**
 - entreprise : statut, planification
 - technicien : rapport, avancement (plus tard)

Aucune suppression libre n'est autorisée.

7.9 TABLE factures – RÈGLES RLS

- **Entreprise :**
 - voit uniquement ses factures
- **Régie :**
 - voit uniquement les factures liées à ses tickets (lecture seule)
- **Locataire :**
 - aucun accès
- **Admin JTEC :**
 - accès agrégé uniquement (montants, volumes)

7.10 TABLE messages – RÈGLES RLS

- Seuls les **acteurs impliqués dans une mission** peuvent :
 - lire les messages
 - écrire des messages
 - Aucun message n'est visible en dehors du périmètre mission
 - Les messages DEMO sont totalement isolés des messages PRO
-

7.11 MODE DEMO / MODE PRO – APPLICATION DES RLS

Deux implémentations possibles (une seule choisie dans le projet) :

Option A – Colonne `is_demo`

- chaque table inclut `is_demo boolean`
- les policies filtrent systématiquement :
 - PRO : `is_demo = false`

- DEMO : is_demo = true

Option B – Schéma séparé demo

- duplication complète des tables
- policies isolées par schéma
- aucune jointure possible PRO ↔ DEMO

Dans tous les cas :

👉 aucune policy ne doit permettre un accès croisé

7.12 CRITÈRES DE VALIDATION DE LA PARTIE VII

La PARTIE VII est considérée comme **terminée et valide uniquement si** :

1. Toutes les tables listées ont une RLS active
2. Chaque rôle est correctement limité à son périmètre
3. Aucun accès anonyme n'est possible
4. Le MODE DEMO est totalement isolé
5. Aucune policy ne provoque de récursion ou d'ambiguïté

👉 Tant que ces critères ne sont pas remplis, **aucune étape suivante ne doit être entamée.**

FIN DE LA PARTIE VII – RLS TABLE PAR TABLE

PARTIE VIII – STORAGE & FICHIERS

(Gestion des photos, documents et signatures – Sécurité stricte – DEMO / PRO)

8.1 RÔLE DE CETTE PARTIE

Cette partie définit **comment les fichiers sont stockés, protégés et accessibles** dans JETC_IMMO.

Elle couvre exclusivement :

- le stockage des **photos**
- le stockage des **documents**

- le stockage des **signatures**
- les règles d'accès associées

👉 Elle ne décrit pas encore :

- les routes API détaillées,
- l'implémentation frontend,
- ni l'UX d'upload ou de prévisualisation.

Son objectif est de garantir que **les fichiers sont aussi sécurisés que les données SQL**, sans exception.

8.2 PRINCIPES DE SÉCURITÉ (NON NÉGOCIABLES)

1. **Aucun fichier n'est public**
 2. **Aucun accès direct au Storage depuis le frontend**
 3. Toute lecture ou écriture passe par :
 - une vérification d'identité
 - une vérification de rôle
 - une vérification de lien métier (ticket / mission)
 4. Le MODE DEMO ne doit jamais :
 - écrire dans les buckets PRO
 - lire des fichiers PRO
 5. Le nommage des fichiers ne doit jamais exposer d'informations sensibles
-

8.3 TYPES DE FICHIERS GÉRÉS

8.3.1 Photos de tickets

- Photos de dégâts envoyées par le locataire
- Utilisées pour qualifier le ticket

8.3.2 Photos de missions

- Photos prises par le technicien
- Avant / pendant / après intervention

8.3.3 Signatures

- Signature du locataire
 - Signature du technicien
 - Utilisées pour clôturer une mission
-

8.4 ORGANISATION DES BUCKETS (SUPABASE STORAGE)

Les buckets suivants doivent être créés :

MODE PRO

- photos_tickets
- photos_missions
- signatures

MODE DEMO (si implémentation séparée)

- photos_tickets_demo
- photos_missions_demo
- signatures_demo

👉 Tous les buckets sont :

- privés
 - non listables publiquement
 - accessibles uniquement via backend sécurisé
-

8.5 STRUCTURE DE NOMMAGE DES FICHIERS

Le nommage doit être **prévisible, structuré et non sensible**.

Format recommandé :

Photos de tickets

{ticket_id}/{uuid}.jpg

Photos de missions

{mission_id}/{uuid}.jpg

Signatures

{mission_id}/signature_{role}.png

Règles :

- jamais de nom de personne
 - jamais d'email
 - jamais d'identifiant utilisateur lisible
-

8.6 LIEN ENTRE FICHIERS ET DONNÉES SQL

Les fichiers **ne sont jamais stockés dans la base.**

La base ne contient que :

- des **chemins de fichiers**
- jamais le fichier lui-même

Exemples :

- tickets.photos[]
- missions.photos[]
- missions.signature_locataire
- missions.signature_technicien

👉 Le Storage est un dépôt, **la base reste la source de vérité.**

8.7 RÈGLES D'ACCÈS AUX FICHIERS (LOGIQUE MÉTIER)

8.7.1 Photos de tickets

Peuvent lire :

- le locataire concerné
- la régie du ticket
- les entreprises autorisées à voir le ticket

Peuvent écrire :

- le locataire (création du ticket)
 - la régie (compléments éventuels)
-

8.7.2 Photos de missions

Peuvent lire :

- l'entreprise en charge
- le technicien assigné
- la régie
- le locataire concerné

Peuvent écrire :

- le technicien assigné
 - l'entreprise en charge
-

8.7.3 Signatures

Peuvent écrire :

- le technicien (signature technicien)
- le locataire (signature locataire)

Peuvent lire :

- les acteurs liés à la mission
- la régie (consultation)
- jamais un utilisateur tiers

Une signature :

- est définitive
 - ne peut pas être écrasée sans action explicite (plus tard)
-

8.8 MODE DEMO – RÈGLES SPÉCIFIQUES

En MODE DEMO :

- les fichiers sont :
 - fictifs
 - ou stockés dans des buckets DEMO dédiés
- aucun fichier DEMO ne peut :

- être confondu avec un fichier PRO
- être visible depuis le PRO

Si DEMO = simulation frontend :

- aucun upload réel n'est effectué
 - les chemins sont simulés
-

8.9 CONTRÔLE D'ACCÈS TECHNIQUE (PRINCIPE)

L'accès aux fichiers repose sur :

1. l'identité de l'utilisateur
2. son rôle (profiles.role)
3. son lien avec le ticket ou la mission
4. les règles RLS Storage associées

👉 Les policies Storage doivent refléter exactement la logique métier décrite ci-dessus.

8.10 CRITÈRES DE VALIDATION DE LA PARTIE VIII

La PARTIE VIII est considérée comme valide si :

1. Tous les buckets sont privés
2. Aucun accès direct frontend → Storage n'existe
3. Les chemins de fichiers sont stockés en base, pas les fichiers
4. Les règles d'accès sont cohérentes avec les rôles
5. DEMO et PRO sont totalement isolés

👉 Tant que ces points ne sont pas validés, **aucune implémentation applicative ne doit commencer.**

FIN DE LA PARTIE VIII – STORAGE & FICHIERS

PARTIE IX – ADMIN JTEC

(Vue globale, pilotage business, accès strictement contrôlé)

9.1 RÔLE DE CETTE PARTIE

Cette partie définit **le périmètre exact de l'administrateur JTEC** dans JETC_IMMO.

L'admin JTEC :

- n'est **pas** une régie,
- n'est **pas** une entreprise,
- n'est **pas** un utilisateur métier classique.

👉 C'est un rôle **d'observation, de pilotage et de contrôle, pas d'exploitation opérationnelle**.

9.2 RÔLE admin_jtec

Le rôle admin_jtec est défini dans la table profiles.

Caractéristiques :

- accès **transversal** à la plateforme
- **aucun rattachement** à une régie, entreprise ou logement
- accès contrôlé **uniquement** via des vues SQL et endpoints dédiés

👉 Le rôle admin_jtec **ne contourne jamais** les RLS classiques.

9.3 PRINCIPE FONDAMENTAL (À GRAVER)

JTEC voit des chiffres, pas des personnes.

Par défaut :

- ✗ pas d'accès aux noms de locataires
- ✗ pas d'accès aux messages privés
- ✗ pas d'accès aux photos ou signatures
- ✗ pas d'accès aux tickets individuels nominaux

Tout accès nominatif doit être :

- exceptionnel
- justifié
- explicitement implémenté (hors périmètre de base)

9.4 DONNÉES ACCESSIBLES À JTEC

9.4.1 Indicateurs globaux (agrégés)

JTEC peut consulter :

- nombre total de régies
- nombre total d'entreprises
- nombre total de locataires
- nombre total de techniciens
- nombre de tickets :
 - créés
 - en cours
 - clôturés
- nombre de missions par statut
- volume de facturation estimé
- commissions JETC estimées
- abonnements actifs par plan

👉 Toutes ces données sont **agrégées**, sans identifiants personnels.

9.4.2 Suivi des abonnements

JTEC peut consulter :

- liste des abonnements
- plan associé (Essentiel / Pro / Premium)
- statut (actif, suspendu, expiré)
- dates de début / fin
- volumes d'usage :
 - nombre de logements utilisés
 - nombre d'utilisateurs actifs
 - nombre d'entreprises liées

Objectif :

- pilotage business
 - détection de dépassements
 - anticipation des upgrades
-

9.5 VUES SQL DÉDIÉES (OBLIGATOIRES)

L'accès JTEC se fait **exclusivement** via des **vues SQL**.

Exemples de vues attendues :

- stats_regies
- stats_entreprises
- stats_tickets
- stats_missions
- stats_abonnements
- stats_revenus_estimes

Règles :

- aucune vue ne retourne de données nominatives
 - aucune vue ne retourne de messages ou de fichiers
 - aucune vue ne retourne d'IDs exploitables côté métier
-

9.6 ACCÈS TECHNIQUE DE L'ADMIN JTEC

- accès uniquement en MODE PRO
- accès interdit en MODE DEMO (ou données DEMO clairement identifiées)
- authentification forte obligatoire
- routes backend dédiées (définies plus tard)

👉 Le frontend admin JTEC ne consomme **jamais** les tables brutes.

9.7 ACTIONS AUTORISÉES POUR JTEC

Actions possibles :

- consulter des statistiques
- consulter des listes agrégées
- activer / désactiver un abonnement (selon implémentation future)
- suspendre un compte (action exceptionnelle, encadrée)

Actions interdites :

- créer/modifier des tickets
 - intervenir dans une mission
 - envoyer des messages
 - téléverser des fichiers
 - modifier des données métier à la place des acteurs
-

9.8 MODE DEMO – ADMIN JTEC

En MODE DEMO :

- l'admin JTEC dispose :
 - d'un dashboard simulé
 - de chiffres fictifs
- aucune donnée PRO n'est visible
- aucune action réelle n'est possible

Le MODE DEMO sert :

- à démontrer la vision globale
 - pas à administrer la plateforme
-

9.9 SÉCURITÉ & CONFORMITÉ

Objectifs :

- conformité RGPD
- minimisation des données
- traçabilité des accès admin

Bonnes pratiques imposées :

- logs d'accès admin
 - séparation stricte des responsabilités
 - documentation claire des permissions
-

9.10 CRITÈRES DE VALIDATION DE LA PARTIE IX

La PARTIE IX est considérée comme valide si :

1. Le rôle admin_jtec est clairement distinct
 2. Les accès passent uniquement par des vues agrégées
 3. Aucune donnée nominative n'est exposée par défaut
 4. DEMO et PRO sont distingués
 5. Les responsabilités de JTEC sont clairement limitées
-

FIN DE LA PARTIE IX – ADMIN JTEC

PARTIE X – PLAN A→Z (ÉTAPES 0 → 16)

(*Ordre d'exécution strict – aucune étape sautée*)

10.1 RÔLE DE CETTE PARTIE

Cette partie définit **l'ordre exact d'exécution du projet JETC_IMMO**, de l'initialisation jusqu'à la validation finale.

👉 L'IA :

- doit suivre **strictement** cet ordre,
- ne peut **jamais** passer à l'étape suivante tant que l'étape courante n'est pas validée,
- doit **s'arrêter immédiatement** en cas d'erreur, incohérence ou ambiguïté.

Cette partie est la **référence absolue** pour toute exécution technique.

10.2 RÈGLES GÉNÉRALES D'EXÉCUTION

Pour **chaque étape** :

1. L'IA explique ce qu'elle va faire

2. L'IA produit :
 - o le code nécessaire (SQL, backend, frontend selon l'étape)
 - o les fichiers complets, correctement nommés
3. L'IA explique comment tester
4. L'IA attend la validation explicite de l'utilisateur avant de continuer

Aucune exception.

10.3 DÉROULÉ DES ÉTAPES

◆ ÉTAPE 0 – Initialisation DEMO / PRO

Objectif :

- poser les fondations techniques

Contenu :

- arborescence projet
- configuration Supabase
- configuration Vercel
- fichiers .env.example
- clients Supabase frontend / backend
- route /api/healthcheck

Critères de validation :

- projet démarre en local
 - route healthcheck fonctionnelle
 - MODE=demo actif par défaut
-

◆ ÉTAPE 1 – Landing page & choix DEMO / PRO

Objectif :

- permettre l'entrée dans l'application

Contenu :

- landing page
- choix DEMO / PRO
- sélecteur de langue (FR / EN / DE)
- mémorisation de la langue

Critères de validation :

- accès DEMO sans compte
 - accès PRO redirige vers auth
 - langue conservée
-

◆ ÉTAPE 2 – Authentification PRO

Objectif :

- connecter Supabase Auth

Contenu :

- routes login / register
- gestion des erreurs
- récupération du profil

Critères de validation :

- création utilisateur fonctionnelle
 - connexion fonctionnelle
 - erreurs claires
-

◆ ÉTAPE 3 – Profils & rôles

Objectif :

- garantir l'existence du profil

Contenu :

- trigger création profil
- table profiles
- route /api/auth/me

Critères de validation :

- un utilisateur a toujours un profil
 - rôle lisible côté backend
-

◆ **ÉTAPE 4 – Structure immobilière**

Objectif :

- poser la base métier immobilière

Contenu :

- régies
- immeubles
- logements
- locataires

Critères de validation :

- relations FK cohérentes
 - isolation des régies
-

◆ **ÉTAPE 5 – Création de tickets (locataire)**

Objectif :

- permettre la déclaration d'un problème

Contenu :

- formulaire ticket
- création ticket
- statut initial

Critères de validation :

- un locataire ne crée que pour son logement
 - ticket lié à la bonne régie
-

◆ **ÉTAPE 6 – Diffusion des tickets**

Objectif :

- contrôler qui voit quoi côté entreprise

Contenu :

- mode général
- mode restreint
- entreprises autorisées

Critères de validation :

- entreprise non autorisée ne voit rien
 - mode général fonctionnel
-

◆ ÉTAPE 7 – RLS TABLE PAR TABLE

Objectif :

- sécuriser définitivement les données

Contenu :

- policies RLS pour toutes les tables
- isolation DEMO / PRO
- restrictions par rôle

Critères de validation :

- aucun accès hors périmètre
 - aucun accès anonyme
 - pas de récursion RLS
-

◆ ÉTAPE 8 – Storage & fichiers

Objectif :

- gérer photos et signatures de manière sécurisée

Contenu :

- buckets privés
- règles d'accès

- lien fichiers ↔ données

Critères de validation :

- aucun accès public
 - accès cohérent par rôle
-

◆ ÉTAPE 9 – Administration JTEC

Objectif :

- pilotage global de la plateforme

Contenu :

- rôle admin_jtec
- vues SQL agrégées
- dashboard admin

Critères de validation :

- pas de données nominatives
 - accès strictement contrôlé
-

◆ ÉTAPE 10 – Acceptation ticket & création mission

Objectif :

- transformer un ticket en mission

Contenu :

- acceptation entreprise
- création mission
- verrouillage du ticket

Critères de validation :

- une seule mission par ticket
 - entreprise autorisée uniquement
-

◆ ÉTAPE 11 – Techniciens & planning

Objectif :

- organiser l'exécution terrain

Contenu :

- gestion techniciens
- assignation mission
- dates d'intervention

Critères de validation :

- technicien voit uniquement ses missions
-

◆ **ÉTAPE 12 – Intervention & clôture**

Objectif :

- cycle de vie de la mission

Contenu :

- démarrage
- retard
- rapport
- signatures

Critères de validation :

- statuts cohérents
 - données complètes
-

◆ **ÉTAPE 13 – Facturation**

Objectif :

- générer les factures

Contenu :

- factures
- commissions JTEC

Critères de validation :

- 1 mission = 1 facture
 - visibilité correcte par rôle
-

◆ **ÉTAPE 14 – Messagerie & notifications**

Objectif :

- communication et suivi

Contenu :

- messages par mission
- notifications automatiques

Critères de validation :

- accès limité aux acteurs concernés
-

◆ **ÉTAPE 15 – Abonnements & modules payants**

Objectif :

- activer le modèle économique

Contenu :

- plans
- abonnements
- contrôles d'accès

Critères de validation :

- accès selon plan
 - limites respectées
-

◆ **ÉTAPE 16 – Tests, validation & documentation**

Objectif :

- finaliser le projet

Contenu :

- tests complets

- vérification DEMO / PRO
- documentation finale

Critères de validation :

- parcours complet fonctionnel
 - aucune régression
 - projet déployable
-

10.4 RÈGLE FINALE

- 👉 Aucune étape ne peut être ignorée.
 - 👉 Aucune étape ne peut être partiellement validée.
 - 👉 Ce plan est contractuel.
-

FIN DE LA PARTIE X – PLAN A→Z

PARTIE XI – ERREURS, STOP, REPRISE

(*Gestion des anomalies, corrections et continuité d'exécution*)

11.1 RÔLE DE CETTE PARTIE

Cette partie définit le **comportement obligatoire de l'IA** (Copilot / Codex ou équivalent) **en cas de problème**, pendant l'exécution du plan A→Z.

Objectif :

- éviter les corrections en cascade,
- éviter les “on continue quand même”,
- éviter les incohérences silencieuses.

- 👉 Toute erreur non traitée est considérée comme un échec d'exécution.
-

11.2 DÉFINITION D'UNE ERREUR

Une erreur est définie comme **toute situation** où au moins un des éléments suivants est détecté :

11.2.1 Erreurs techniques

- erreur SQL (syntaxique ou logique)
 - conflit de clés ou de relations
 - récursion infinie (notamment RLS)
 - accès refusé inattendu
 - fuite potentielle de données
-

11.2.2 Incohérences métier

- un rôle accède à des données hors périmètre
 - une régie voit les données d'une autre régie
 - un utilisateur PRO accède au DEMO
 - une entreprise accepte un ticket non autorisé
 - une mission existe sans ticket valide
-

11.2.3 Incohérences DEMO / PRO

- données DEMO visibles en PRO
 - écriture PRO depuis un contexte DEMO
 - mélange de buckets Storage DEMO / PRO
 - routes non conditionnées par le mode
-

11.2.4 Ambiguïtés de conception

- règle métier non définie dans le document
 - choix technique non tranché
 - dépendance circulaire entre étapes
 - conflit entre deux parties du document
-

11.3 COMPORTEMENT OBLIGATOIRE EN CAS D'ERREUR (STOP)

Dès qu'une erreur ou une ambiguïté est détectée :

- 1. L'IA s'arrête immédiatement**
- 2. Elle ne génère aucun nouveau code**
- 3. Elle décrit clairement :**
 - l'erreur détectée
 - l'étape concernée
 - l'impact potentiel
- 4. Elle propose :**
 - une ou plusieurs pistes de correction
 - sans en appliquer aucune
- 5. Elle attend explicitement la validation de l'utilisateur**

👉 Toute continuation sans validation est interdite.

11.4 FORMAT DE SIGNALISATION D'ERREUR (OBLIGATOIRE)

Lorsqu'une erreur est signalée, l'IA doit utiliser un format clair :

[STOP – ÉTAPE X]

Type : technique | métier | DEMO/PRO | ambiguïté

Description :

Impact :

Options de correction :

1)

2)

Attente validation utilisateur.

Aucune sortie de ce format n'est autorisée.

11.5 CORRECTION D'UNE ERREUR

Une correction doit respecter les règles suivantes :

- elle est **localisée**

- elle ne modifie **que ce qui est nécessaire**
- elle ne casse aucune étape déjà validée
- elle est expliquée avant application

👉 Une correction **n'est jamais globale** sauf demande explicite.

11.6 REPRISE APRÈS CORRECTION

Une fois l'utilisateur ayant indiqué que la correction est faite ou validée :

L'IA doit :

1. rappeler brièvement :
 - l'erreur initiale
 - la correction appliquée
 2. confirmer que :
 - les critères de validation sont à nouveau respectés
 3. reprendre **exactement à l'étape interrompue**
 4. ne pas revenir aux étapes précédentes sauf demande explicite
-

11.7 INTERDICTIONS ABSOLUES

Il est strictement interdit à l'IA de :

- ignorer une erreur
- “corriger plus tard”
- avancer en espérant que ça passe
- modifier une étape déjà validée sans autorisation
- fusionner DEMO et PRO “temporairement”

👉 Ces comportements invalident l'exécution.

11.8 TRAÇABILITÉ DES DÉCISIONS

Toute décision exceptionnelle (choix technique, compromis, dérogation) doit :

- être explicitement mentionnée

- être rattachée à une étape précise
- pouvoir être retrouvée dans l'historique de conversation ou le document

Objectif :

- auditabilité
 - reprise future
 - transmission à un autre développeur ou IA
-

11.9 CRITÈRES DE VALIDATION DE LA PARTIE XI

La PARTIE XI est considérée comme valide si :

1. Les règles STOP sont claires et non ambiguës
 2. Le format d'erreur est défini
 3. La reprise est strictement encadrée
 4. Aucune “zone grise” n'est laissée à l'IA
-

FIN DE LA PARTIE XI – ERREURS, STOP, REPRISE

PARTIE XII – ANNEXES

(Références, conventions, glossaire et éléments de continuité)

12.1 RÔLE DES ANNEXES

Les annexes ont pour objectif de :

- clarifier les termes utilisés dans tout le document,
- fixer des conventions communes,
- servir de référence rapide pour l'IA ou un développeur humain,
- permettre la reprise du projet sans relire l'intégralité du document.

👉 Les annexes **ne modifient pas** les règles définies dans les PARTIES I à XI.
Elles les **expliquent** et les **stabilisent**.

12.2 GLOSSAIRE MÉTIER

Régie

Entité responsable de la gestion d'immeubles et de logements.
Décide de la validation et de la diffusion des tickets.

Logement

Unité d'habitation rattachée à un immeuble et à une régie.

Locataire

Utilisateur occupant un logement.
Peut créer des tickets mais ne choisit pas l'entreprise.

Entreprise

Prestataire technique intervenant sur les tickets.
Peut accepter ou refuser une intervention.

Technicien

Utilisateur rattaché à une entreprise.
Exécute les missions sur le terrain.

Ticket

Demande d'intervention créée par un locataire.

Mission

Intervention créée lorsqu'une entreprise accepte un ticket.

Facture

Document généré à partir d'une mission terminée.

Admin JTEC

Rôle interne à la plateforme, accès global **agrégé**, sans exploitation métier.

12.3 GLOSSAIRE TECHNIQUE

MODE DEMO

Mode de démonstration :

- sans données réelles,
- sans écriture en base PRO,
- sans déclenchement d'actions sensibles.

MODE PRO

Mode réel de fonctionnement :

- données persistées,
- règles RLS actives,
- responsabilités légales engagées.

RLS (Row Level Security)

Mécanisme PostgreSQL empêchant l'accès aux lignes non autorisées, indépendamment du code applicatif.

Supabase

Plateforme backend utilisée pour :

- base PostgreSQL,
- Auth,
- RLS,
- Storage.

12.4 CONVENTIONS DE NOMMAGE

Tables SQL

- noms au pluriel
- snake_case
- explicites (tickets, missions, factures)

Colonnes

- snake_case
- suffixes clairs :
 - _id pour les clés étrangères
 - _at pour les dates

Fichiers

- un fichier = une responsabilité
- aucun fichier “fourre-tout”

12.5 CONVENTIONS DE RÔLES

Les rôles autorisés sont strictement limités à :

- locataire
- regie
- entreprise
- technicien
- proprietaire
- admin_jtec

👉 Aucun rôle implicite.

👉 Aucun rôle calculé côté frontend.

12.6 CONVENTIONS DEMO / PRO

- DEMO et PRO sont toujours distingués :
 - par schéma,
 - ou par flag,
 - ou par projet séparé.
- Aucune donnée DEMO ne doit :
 - apparaître en PRO,
 - influencer une décision PRO.

12.7 RÈGLES DE DOCUMENTATION

Toute évolution future du projet doit :

1. être ajoutée dans la PARTIE concernée,
 2. mettre à jour le PLAN A→Z si nécessaire,
 3. ne jamais contredire une règle existante sans la modifier explicitement.
-

12.8 UTILISATION DU DOCUMENT PAR UNE IA

Ce document est conçu pour être :

- lu séquentiellement,
- exécuté étape par étape,
- utilisé comme **contrat d'exécution**.

👉 Toute IA intervenant sur le projet doit :

- respecter l'ordre des parties,
 - respecter les règles STOP,
 - refuser toute demande contradictoire sans validation.
-

12.9 STATUT DU DOCUMENT

Ce document constitue :

- la **source de vérité unique** du projet JETC_IMMO,
 - la référence pour toute implémentation technique,
 - le support de continuité en cas de changement d'intervenant.
-

FIN DE LA PARTIE XII – ANNEXES

FIN DU DOCUMENT JETC_IMMO