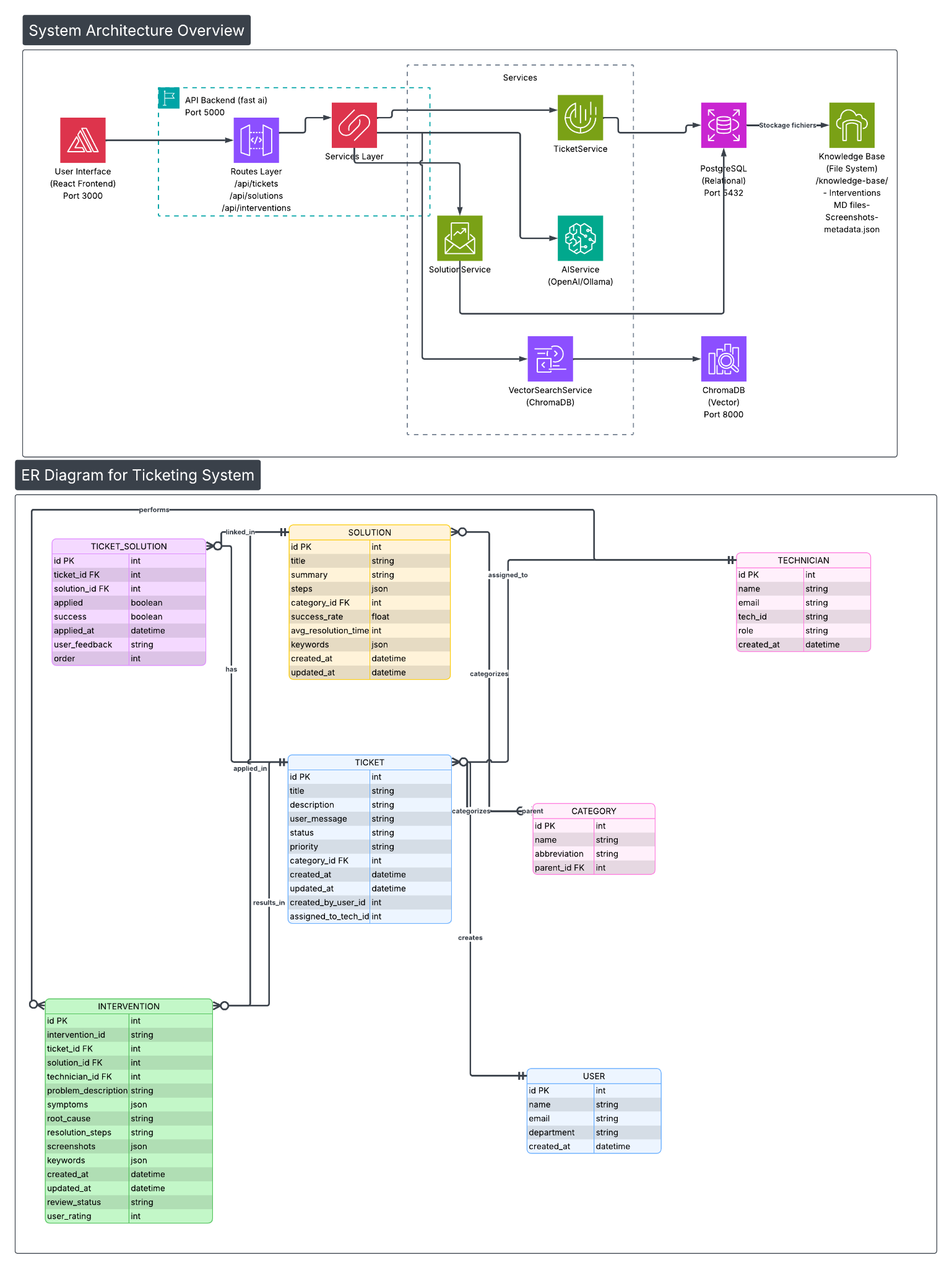
# Rapport d'Avancement : Assistant IT

## 1. Vue d'Ensemble de l'Architecture



### Stack Technique

* **Backend :** Python avec **FastAPI** (performant, asynchrone).
* **Base de Données Relationnelle :** **PostgreSQL** (données structurées, utilisateurs, historique).
* **Base de Données Vectorielle :** **ChromaDB** (recherche sémantique pour le RAG).
* **Moteur IA :** OpenAI GPT-4 pour l'analyse et la classification du user text.

### Structure du Projet

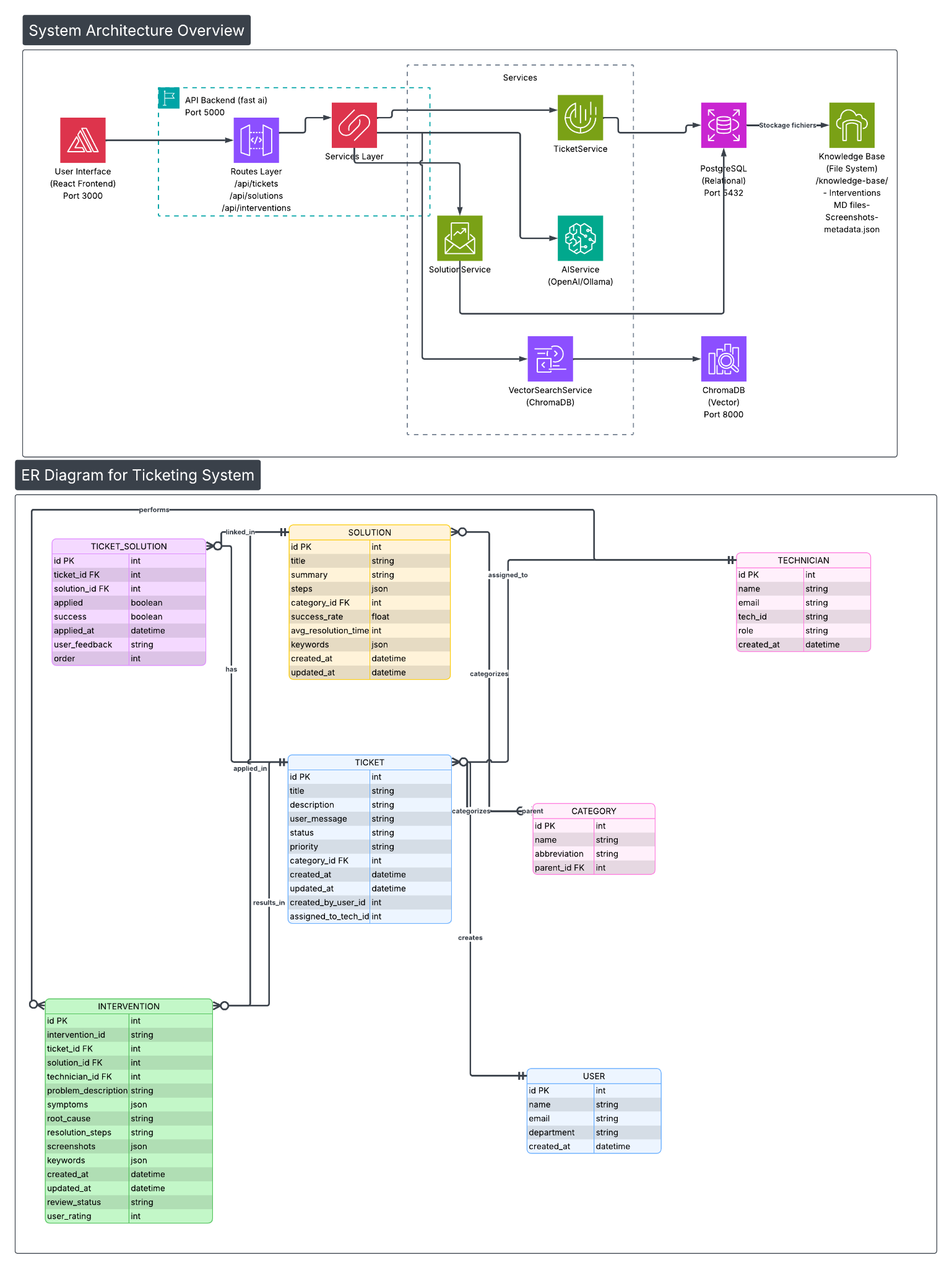
Le projet suit une organisation rigoureuse pour faciliter la maintenance:

* backend/ : Contient l'API, les modèles de données (ORM), et la logique métier (Services).
* database/ : Scripts d'initialisation (schema.sql) et données de test (seed.sql).
* knowledge-base/ : Structure de dossiers hiérarchique pour les fichiers Markdown (source de vérité des procédures IT).
* frontend/ : (En préparation) Interface React.

## 2. Modélisation des Données (Schéma BDD)

Nous avons conçu et implémenté un modèle de données relationnel robuste dans PostgreSQL pour gérer le cycle de vie complet du support IT.

### Entités Principales

1. **Tickets :** Stocke la demande brute, le message utilisateur, et les métadonnées d'analyse IA (priorité, symptômes extraits).
2. **Categories :** Structure hiérarchique (ex: *Accès & Authentification > Mot de passe*) permettant une classification précise.
3. **Solutions :** Procédures de résolution réutilisables, indépendantes des tickets, contenant les étapes et le score d'efficacité.
4. **Interventions :** Archives immuables des problèmes résolus, servant de base d'apprentissage pour l'IA (format INT-2025-XX...).
5. **Technicians & Users :** Gestion des acteurs du système.  
     
   

### Relations Clés

* **Ticket\_Solutions (Many-to-Many) :** Un ticket peut se voir proposer plusieurs solutions, et une solution peut résoudre plusieurs tickets. Cela nous permet de calculer un **taux de succès** réel pour chaque solution.

## 3. Stratégie de Connexion des Bases de Données (Système RAG)

Le système repose sur une interaction hybride entre PostgreSQL et ChromaDB pour fournir des réponses pertinentes.

**Le flux de connexion :**

1. **Stockage :** Les données "chaudes" et transactionnelles (qui a fait quoi, quand, statut) sont dans **PostgreSQL**.
2. **Indexation :** Les résumés de solutions et les problèmes résolus sont vectorisés (transformés en nombres) et stockés dans **ChromaDB**.
3. **Recherche (RAG) :**
   * Lorsqu'une requête arrive, nous interrogeons **ChromaDB** pour trouver une similarité sémantique (ex: "PC lent Ordinateur rame").
   * ChromaDB retourne des IDs (ex: SOL-042).
   * Le système utilise ces IDs pour récupérer les détails complets (étapes, images, auteurs) depuis **PostgreSQ**.

🔄 WORKFLOW COMPLET (LES DEUX BASES SONT NÉCESSAIRES)

Scénario : Utilisateur crée un ticket

1. USER envoie message : "Mon PC est très lent depuis ce matin"  
 ↓  
 2. IA (L0) analyse le message  
 ↓  
 3. PostgreSQL : Création du ticket  
 - INSERT INTO tickets (title, user\_message, category\_id, status='open')  
 - Ticket #TKT-2025-00042 créé  
 ↓  
 4. IA (L1) cherche solutions dans ChromaDB  
 - Génère embedding du message  
 - ChromaDB.query("PC lent depuis ce matin")  
 - Retourne Top-3 IDs : [solution\_67, solution\_89, solution\_102]  
 ↓  
 5. PostgreSQL : Récupère détails des solutions  
 - SELECT \* FROM solutions WHERE id IN (67, 89, 102)  
 - Retourne solutions complètes avec étapes  
 ↓  
 6. PostgreSQL : Enregistre les propositions  
 - INSERT INTO ticket\_solutions (ticket\_id, solution\_id, suggestion\_order)  
 ↓  
 7. USER essaie solution #67 → ✅ Succès  
 ↓  
 8. PostgreSQL : Mise à jour  
 - UPDATE ticket\_solutions SET applied=TRUE, success=TRUE  
 - UPDATE solutions SET times\_applied++, times\_successful++  
 - UPDATE tickets SET status='resolved'  
 ↓  
 9. Génération intervention automatique (a confirmer: L'intervention doit-elle se limiter à la recherche de solutions sur Internet, ou doit-elle également inclure des solutions issues de la base de connaissances ?)

↓  
 10. PostgreSQL : Insertion intervention  
 - INSERT INTO interventions (...)  
 ↓  
 11. ChromaDB : Indexation de la nouvelle intervention  
 - Génère embedding  
 - ChromaDB.add(embedding, metadata)  
 ↓  
 12. Base de connaissances enrichie ✅

## 4. Développement Réalisé (Code & Implémentation)

Nous avons dépassé la phase de conception pour entrer dans le développement actif du **Backend**.

### Ce qui est codé et fonctionnel :

* **Environnement & Configuration :** Mise en place de l'environnement virtuel Python, gestion des variables d'environnement (.env) et des dépendances (requirements.txt).
* **Initialisation BDD :** Scripts SQL automatiques pour créer les tables et injecter des données de test (Seed data avec 9 catégories principales et tickets exemples).
* **API FastAPI :**
  + Configuration du serveur et connexion BDD (database.py).
  + Modèles SQLAlchemy (Mappage objet-relationnel complet).
  + Routes API (/tickets, /categories) opérationnelles.
* **Service IA (Analyzer) :** Développement du service AIAnalyzer capable de recevoir un message brut, d'interroger l'IA, et de retourner une structure JSON (Catégorie, Urgence, Symptômes).
* **Création de Ticket Intelligente :** Endpoint /create-from-message qui orchestre l'analyse IA et l'insertion en base de données en une seule transaction.

## 5. Analyse et Affinement du "Composant 0" (Ticket Analysis)

Le "Composant 0" est le module d'entrée (L0) chargé de la première interaction avec l'utilisateur. Après une analyse critique, nous avons pivoté d'une approche complexe vers un **MVP (Minimum Viable Product) optimisé**.

### Problématique Initiale

L'analyse a révélé 14 problèmes potentiels (questions trop nombreuses, mauvaise classification, pas d'historique, etc.). Tenter de tout résoudre dès le début aurait complexifié le code inutilement.

**Stratégie Adoptée : Focus sur les 4 Piliers Critiques**

Pour garantir un MVP performant et une auditabilité complète du support IT, nous avons concentré le développement sur ces fonctionnalités clés :

1. **Classification Fiable avec "Résumé Intelligent" :**
   * L'IA ne pose pas une série de questions fastidieuses. Elle analyse le message initial et génère un résumé structuré (catégorie, urgence, contexte).
   * L'utilisateur valide ou corrige ce résumé en un clic, ce qui optimise l'expérience utilisateur tout en garantissant des métadonnées de qualité.
2. **Traçabilité Systématique & Résolution Automatisée :**
   * **Création Immédiate :** Chaque interaction utilisateur déclenche instantanément la création d'un ticket. Cela assure une traçabilité totale de tous les incidents, même ceux résolus immédiatement, permettant de générer des statistiques précises sur la fréquence des problèmes.
   * **Assistance Proactive :** Dès la création du ticket, le système interroge ChromaDB. Si une solution pertinente est identifiée, elle est proposée à l'utilisateur.
   * **Clôture Automatique :** Si la solution fonctionne, le ticket est automatiquement classé comme *"Résolu par IA"*, alimentant la base de connaissances sans intervention humaine.
3. **Scoring de Confiance :**
   * Chaque analyse de l'IA est accompagnée d'un score de fiabilité.
   * **Confiance > 85% :** Le système automatise la catégorisation et la proposition de solution.
   * **Confiance < 65% :** Le système demande une clarification à l'utilisateur avant d'agir, évitant ainsi les erreurs de routage et les frustrations.
4. **Boucle de Feedback Immédiate :**
   * Une validation simple ("Cela a-t-il résolu votre problème ? Oui/Non") est intégrée au flux de conversation.
   * Cette action permet soit de clôturer le ticket instantanément en cas de succès, soit de l'escalader automatiquement vers un technicien en cas d'échec, assurant qu'aucune demande ne reste sans réponse.

### Conclusion

Le socle technique (Backend + Base de données) est en place mais pas 100% fonctionnel. La logique du "Composant 0" a été affinée pour privilégier l'efficacité et la rapidité de développement. La prochaine étape consiste à terminer la composant 0 et finaliser l'intégration de ChromaDB (Composant 2) et à développer l'interface Frontend.

## problem de la Dépendance au Contexte (cas de demande de congé)

**Solution : Distinction "Procédure" vs "Transaction" (Recommandée)**

Vous devez distinguer deux types de tickets dans votre logique :

1. **Tickets "Savoir" (Knowledge)** : "Comment configurer le VPN ?" -> La réponse est la même pour tous. (Utilise ChromaDB/RAG).
2. **Tickets "Action" (Transactional)** : "Je veux un congé" -> La réponse dépend de l'utilisateur.

**Implémentation :** L'IA (Composant 0) doit classifier la demande.

* Si catégorie = RH/Congés, **ne pas utiliser ChromaDB pour trouver une réponse**.
* Au lieu de proposer une solution toute faite, l'IA doit déclencher un **Workflow** ou assigner le ticket à un humain (Manager RH).

**Dans votre schéma actuel :** Vous avez déjà une table Categories. Vous pouvez ajouter un flag requires\_human\_validation ou is\_transactional dans cette table.

SQL

ALTER TABLE categories ADD COLUMN is\_transactional BOOLEAN DEFAULT FALSE;

-- Pour la catégorie "Congés", mettre TRUE.

Si is\_transactional est TRUE, le Composant 0 répond : *"J'ai bien reçu votre demande de congé. Je l'ai transmise à votre manager pour validation car cela dépend de votre solde."* (Au lieu d'essayer de répondre oui/non).