Documentation Complète – Jour 1 : Fondation du Projet Backend

Last edited	@July 26, 2025 2:19 PM
Tags	

Objectif du Jour 1

L'objectif principal du Jour 1 était de construire une **fondation de développement de qualité professionnelle** pour le backend de l'application. Il ne s'agissait pas d'écrire de la logique métier, mais de mettre en place tous les outils et infrastructures nécessaires pour garantir que le développement futur soit **stable, sécurisé, cohérent et efficace**.

À la fin de cette journée, nous avons un environnement où chaque pièce a une raison d'être, depuis la gestion des services jusqu'à la qualité du code.

---

1. L'Infrastructure : Docker et les Services Conteneurisés

Ce que nous avons utilisé :

- **Docker Desktop :** Le moteur pour faire tourner des conteneurs.
- **Docker Compose :** L'outil d'orchestration pour définir et lancer notre application multi-services.
- **Fichiers créés :** docker-compose.yml, Dockerfile, .env (à la racine).

Pourquoi nous l'avons utilisé :

L'utilisation de Docker et Docker Compose est au cœur de notre stratégie pour trois raisons fondamentales :

1. **Reproductibilité (Reproducibility) :** En définissant notre base de données (SQL Server), notre antivirus (ClamAV) et nos deux API dans un fichier docker-compose.yml, nous garantissons que **n'importe quel développeur (ou un serveur de production) peut lancer l'exact même environnement**

avec une seule commande (docker compose up). Cela élimine le syndrome du "ça marche sur ma machine".

2. **Isolation (Isolation) :** Chaque service tourne dans son propre conteneur, isolé des autres et du système d'exploitation de la machine. Nous n'avons pas besoin d'installer SQL Server ou ClamAV directement sur notre ordinateur, ce qui évite les conflits de versions, les problèmes de ports et garde notre machine propre.
3. **Gestion de l'Environnement (Environment Management) :** Le fichier .env à la racine est devenu la **source de vérité unique** pour la configuration de tout l'environnement. Docker Compose injecte les bonnes variables (comme les DATABASE_URL) dans les bons conteneurs, ce qui nous permet de gérer la configuration de manière centralisée et sécurisée (le fichier .env n'est jamais commité sur Git).

---

2. La Base de Données : Prisma, le Schéma comme Source de Vérité

Ce que nous avons utilisé :

- **Prisma CLI :** L'outil en ligne de commande pour gérer notre base de données.
- **Prisma Client :** Le client TypeScript qui nous permettra d'interroger la base de données de manière sécurisée.
- **Fichiers créés :** app-backend/prisma/schema.prisma, app-backend/prisma/seed.ts, app-backend/.env.

Pourquoi nous l'avons utilisé :

Prisma est bien plus qu'un simple connecteur de base de données. C'est un **ORM (Object-Relational Mapper)** de nouvelle génération qui nous apporte trois avantages majeurs :

1. **Le Schéma comme Source de Vérité (Schema as a Single Source of Truth) :** Le fichier schema.prisma est devenu la définition **lisible par un humain** de notre base de données. Au lieu de manipuler des scripts SQL complexes, nous décrivons nos tables et leurs relations dans ce fichier unique. C'est plus simple, moins sujet aux erreurs et versionnable avec Git.

2. **Migrations Sûres et Reproductibles (Safe Migrations) :** La commande npx prisma migrate dev est notre filet de sécurité. Elle compare notre schema.prisma à la base de données et **génère automatiquement le script SQL** nécessaire pour les synchroniser. Ce script est sauvegardé, ce qui garantit que nous pouvons recréer la base de données à l'identique n'importe où, n'importe quand.
3. **Auto-complétion et Sécurité des Types (Type Safety) :** Après chaque migration, Prisma génère un **client TypeScript** (@prisma/client) qui est **parfaitement adapté à notre schéma**. Quand nous écrirons prisma.plainte.create(...), TypeScript saura exactement quels champs sont obligatoires, leurs types, etc. Cela élimine une classe entière de bugs et rend le développement beaucoup plus rapide.

---

3. L'Hygiène du Projet : ESLint, le Gardien de la Qualité

Ce que nous avons utilisé :

- **ESLint v9 :** La dernière version de l'outil de "linting" le plus populaire en JavaScript/TypeScript.
- **typescript-eslint :** Le plugin qui permet à ESLint de comprendre les spécificités de TypeScript.
- **Fichier créé :** app-backend/eslint.config.js.

Pourquoi nous l'avons utilisé :

Mettre en place ESLint dès le premier jour est un investissement qui paie tout au long du projet.

1. **Maintenir la Cohérence du Code :** ESLint agit comme un arbitre impartial qui force tout le code du projet à respecter le même style et les mêmes conventions. Cela rend le code plus facile à lire et à comprendre pour n'importe quel développeur.
2. **Prévenir les Erreurs Logiques :** ESLint est configuré pour détecter des schémas de code qui, bien que syntaxiquement corrects, sont souvent des sources de bugs (par exemple, des variables déclarées mais jamais utilisées). Il nous aide à écrire un code plus robuste.
3. **Automatiser la Relecture :** En exécutant npm run lint, nous automatisons la première passe d'une relecture de code. Nous n'avons plus à nous soucier

des détails de formatage et pouvons nous concentrer sur la logique métier. C'est une étape essentielle pour le travail en équipe et l'intégration continue (CI).

Checklist Finale et Détaillée du Jour 1

- [✔️] **Infrastructure Docker configurée** avec un docker-compose.yml orchestrant 4 services (db, clamav, lookup-api, trx-api).
- [✔️] **Variables d'environnement centralisées** dans un fichier .env racine, avec une version locale dans app-backend/ pour les commandes CLI.
- [✔️] **Environnement Docker lancé et validé** : tous les conteneurs sont running et sqlserver-dev est Healthy.
- [✔️] **Prisma installé et initialisé** dans le projet app-backend.
- [✔️] **Schéma de base de données complet défini** dans prisma/schema.prisma, en résolvant tous les problèmes de compatibilité avec SQL Server (types natifs, cascades, relations).
- [✔️] **Migration initiale créée et appliquée** avec prisma migrate dev, créant la base ComplaintDev et ses 14 tables.
- [✔️] **Script de "seeding" créé et exécuté** avec prisma db seed pour peupler les tables de référence.
- [✔️] **Base de données validée visuellement** avec prisma studio, confirmant la présence des tables et des données de seed.
- [✔️] **ESLint v9 configuré et validé** avec un fichier eslint.config.js fonctionnel. La commande npm run lint s'exécute sans erreur.
- [✔️] **Premier commit de sauvegarde effectué**, marquant la fin de la mise en place de la fondation du projet.