Rapport de Projet : Application de Gestion des Dons de Sang

Table des Matières

1. Introduction

1.1 Sujet Choisi

Ce projet de fin d'études porte sur le développement d'une application web REST monolithique dédiée à la **gestion des dons de sang**. Le don de sang est un acte civique essentiel, mais sa coordination peut s'avérer complexe. L'application vise à moderniser et simplifier ce processus crucial.

1.2 Problème Résolu

L'objectif principal est de résoudre les défis liés à la coordination des dons et des demandes de sang, qui incluent souvent :

- **Difficulté pour les donneurs de proposer leur sang :** Manque de plateformes centralisées pour signaler leur disponibilité.
- **Complexité pour les demandeurs de trouver le sang nécessaire :** Surtout pour les groupes sanguins rares ou en cas d'urgence.
- **Manque de visibilité sur les besoins et les disponibilités :** Les centres de don peuvent manquer d'informations en temps réel sur les stocks et les donneurs potentiels.
- **Processus d'affectation manuel et inefficace :** L'association d'une demande à un donneur est souvent fastidieuse.

Notre solution propose une plateforme centralisée où les donneurs potentiels peuvent s'enregistrer et proposer leur sang, et où les demandes de sang peuvent être soumises. Un administrateur joue un rôle clé en facilitant l'association de ces

demandes avec les propositions, optimisant ainsi la distribution du sang.

1.3 Équipe du Projet et Rôles

Ce projet a été réalisé en équipe, avec une répartition des rôles et responsabilités comme suit :

- **Mohamed El Amine Kharraz :** Responsable du **Backend**. Implémentation de l'API RESTful avec FastAPI, gestion de la base de données MySQL via SQLAlchemy, authentification, et logique métier principale.
- **Mohamed El Mehdi Zoubae :** Responsable du **Frontend**. Développement de l'interface utilisateur avec React.js, intégration des styles Tailwind CSS, gestion de l'état (authentification, navigation) et interaction avec l'API backend.
- **Younes Nouaiti :** Responsable **DevOps**. Mise en place de l'infrastructure de déploiement (conteneurisation avec Docker) et automatisation des pipelines d'intégration continue et de déploiement continu (CI/CD avec GitHub Actions).

2. Fonctionnalités Implémentées

L'application est divisée en deux composants principaux, le backend (API) et le frontend (interface utilisateur), fonctionnant de manière monolithique.

2.1 Backend (FastAPI)

Le backend a été développé avec FastAPI pour sa rapidité et sa capacité à générer des API REST robustes.

- **Gestion des Utilisateurs :**
  - Endpoint d'inscription (POST /utilisateurs/) pour permettre aux nouveaux utilisateurs de créer un compte avec un rôle par défaut (normal). Les mots de passe sont hachés de manière sécurisée.
  - Endpoint de connexion (POST /token) qui authentifie les utilisateurs et renvoie des informations clés (ID, email, rôle) pour la gestion de session côté frontend.
- **Gestion des Groupes Sanguins :**
  - Endpoints CRUD (GET, POST) pour gérer les types de groupes sanguins disponibles dans le système.
- **Gestion des Propositions de Don :**
  - Endpoint (POST /propositionsdon/) permettant à un utilisateur de déclarer sa disponibilité pour un don, en spécifiant la date, la localisation, etc.
  - Endpoint (GET /propositionsdon/) pour consulter les propositions (filtrées par statut 'en attente' pour l'administration).
- **Gestion des Demandes de Don :**

    - Endpoint (POST /demandesdon/) pour soumettre une demande de sang, précisant le groupe sanguin requis, la quantité, l'urgence et une description.
    - Endpoint (GET /demandesdon/) pour consulter les demandes (filtrées par statut 'en attente' pour l'administration).
- **Gestion des Affectations de Don (Fonctionnalité clé Admin) :**
    - Endpoint (POST /affectationsdon/) permettant aux administrateurs d'associer une proposition de don à une demande de don spécifique. Cette action met à jour les statuts des entités concernées.
    - Endpoint (GET /affectationsdon/) pour consulter l'historique de toutes les affectations.
- **Statistiques Générales (Fonctionnalité clé Admin) :**
    - Endpoint (GET /stats/) fournissant des chiffres agrégés sur le nombre total d'utilisateurs, de propositions, de demandes et d'affectations, ainsi que leurs statuts.
- **Sécurité et Cohérence :**
    - Implémentation de CORS pour la communication inter-origines avec le frontend.
    - Validation des données entrantes via Pydantic.
    - Gestion basique des autorisations (accès admin restreint à certaines routes).

2.2 Frontend (React.js)

Le frontend offre une interface utilisateur dynamique et réactive, construite avec React.js et stylisée avec Tailwind CSS.

- **Page de Connexion :** Interface intuitive pour l'authentification des utilisateurs, avec gestion des erreurs et communication avec l'API backend.
- **Page d'Inscription :** Formulaire multi-étapes pour permettre aux nouveaux utilisateurs de créer un compte, intégrant la validation des champs et la soumission sécurisée des données.
- **Navigation Conditionnelle :** Un Header cohérent qui s'adapte en fonction du statut de connexion et du rôle de l'utilisateur (normal ou admin).
- **Tableaux de Bord :**
    - Un tableau de bord utilisateur (normal) avec des options pour proposer ou demander un don.
    - Un tableau de bord administrateur (admin) avec des liens vers la gestion des affectations et les statistiques.
- **Formulaires Spécifiques :**
    - Formulaire "Proposer un Don" : Interface pour les donneurs pour enregistrer leur disponibilité et leurs détails de don.
    - Formulaire "Faire une Demande de Don" : Interface pour soumettre des

demandes, avec un sélecteur de groupe sanguin dynamique.
- **Pages Administratives :**
  - "Gérer les Affectations" : Affiche les propositions et demandes en attente et permet à l'administrateur de les lier.
  - "Statistiques" : Présente les données globales de l'application de manière claire et concise.
- **Gestion d'État :** Utilisation du Contexte React (AuthContext) pour gérer globalement l'état d'authentification et les informations de l'utilisateur connecté.

3. Analyse des Défis et Solutions Apportées

Le développement de cette application a présenté plusieurs défis, dont la résolution a permis de renforcer la robustesse et la compréhension des technologies utilisées.

3.1 Défis Techniques

- **Intégration FastAPI et SQLAlchemy :** Assurer un mapping correct entre les modèles SQLAlchemy (ORM) et les schémas Pydantic pour la validation et la sérialisation des données. Les erreurs d'indentation et les typos ont parfois compliqué le démarrage du serveur.
- **Configuration Frontend avec Vite et Tailwind CSS :** L'intégration de Tailwind CSS avec Vite a posé des défis inattendus, notamment liés à la détection des fichiers de configuration (tailwind.config.js, postcss.config.js) et à la compatibilité des plugins PostCSS (@tailwindcss/vite, @tailwindcss/postcss).
- **Communication Frontend-Backend (CORS) :** Le blocage des requêtes inter-origines par les navigateurs a nécessité la mise en place de middleware CORS dans FastAPI pour autoriser les requêtes provenant du frontend.
- **Gestion des Sessions et Autorisations (Simplifiée) :** Implémenter une logique d'authentification et d'autorisation qui soit fonctionnelle pour la démonstration tout en reconnaissant les meilleures pratiques (JWT) pour une application de production.

3.2 Défis de Développement et d'Intégration

- **Débogage des Erreurs Inattendues :** Des erreurs apparemment mineures (point-virgule mal placé, erreurs de frappe, problèmes de cache Node.js/Vite) ont parfois conduit à des blocages frustrants, nécessitant une analyse minutieuse des messages d'erreur et une approche méthodique.
- **Cohérence des Données :** S'assurer que les données créées via les différents formulaires du frontend (inscription, proposition, demande) sont correctement envoyées, validées et stockées dans le backend.
- **Environnements de Développement :** Les différences de comportement entre

les systèmes d'exploitation (sensibilité à la casse sur les chemins de fichiers, politiques d'exécution PowerShell) ont nécessité des ajustements spécifiques.

3.3 Solutions Mises en Œuvre

- **Approche Pas-à-Pas et Itérative :** Le développement a été mené par petites étapes, testant chaque fonctionnalité ou configuration avant de passer à la suivante. Cela a permis d'isoler rapidement les problèmes.
- **Vérification de la Chaîne Complète :** En cas d'erreur (Failed to fetch, N/A dans l'affichage), la démarche a consisté à vérifier la chaîne de communication : du navigateur (console, outils réseau) au frontend, puis à l'API backend (Swagger UI), et enfin à la base de données (requêtes SQL).
- **Réinitialisation et Nettoyage de l'Environnement :** Pour les problèmes persistants (style Tailwind, erreurs uvicorn), des réinitialisations complètes des environnements virtuels, des dossiers node_modules et des fichiers de cache ont souvent été nécessaires pour repartir sur une base propre.
- **Mise à Jour des Schémas API :** L'ajustement précis des schémas Pydantic du backend (AffectationDonResponse, réponse de /token) a été crucial pour que le frontend reçoive toutes les données nécessaires à son affichage.
- **Validation Frontend Améliorée :** L'implémentation de validateStep1() dans le formulaire d'inscription a permis de fournir un feedback immédiat à l'utilisateur, améliorant l'expérience.
- **Utilisation des Outils de Débogage :** La console du navigateur, les messages d'erreur de terminal d'Uvicorn/Vite et les outils de développement (pour voir les requêtes réseau) ont été indispensables pour le diagnostic.

4. Conclusion et Perspectives

Ce projet a permis de concevoir, développer et déployer une application web REST monolithique fonctionnelle pour la gestion des dons de sang. Il démontre une solide compréhension des concepts de développement full-stack moderne, incluant la conception de bases de données, le développement d'API robustes, la création d'interfaces utilisateur réactives, et la gestion de l'état.

Les défis rencontrés, particulièrement ceux liés à l'intégration des différentes technologies, ont été des opportunités d'apprentissage précieuses, renforçant les compétences en débogage et en résolution de problèmes.

Perspectives d'amélioration future :

- **Implémentation Complète de JWT :** Renforcer la sécurité par l'utilisation de jetons JWT chiffrés et la protection de toutes les routes de l'API nécessitant une authentification.

- **Fonctionnalités CRUD Complètes :** Ajouter des interfaces utilisateur pour les opérations de mise à jour et de suppression (PUT/PATCH, DELETE) pour toutes les entités.
- **Notifications en Temps Réel :** Implémenter des notifications pour les donneurs et demandeurs (ex: affectation réussie, rappel de don).
- **Géolocalisation Avancée :** Utiliser des APIs de cartographie pour visualiser les propositions et demandes sur une carte.
- **Historique Détaillé pour l'Utilisateur :** Permettre aux utilisateurs de consulter un historique plus détaillé de leurs dons et demandes.
- **Docker et CI/CD :** Finaliser les configurations Docker et les pipelines GitHub Actions pour un déploiement automatisé en production.
- **Tests Automatisés :** Étendre la couverture des tests unitaires et ajouter des tests d'intégration.

Ce projet constitue une base solide pour une application potentiellement utile dans le monde réel, démontrant les compétences techniques acquises tout au long de la formation.