Architecture du Système Basé sur Microservices

1 Introduction

Le système décrit est une application web conçue pour deux types d'utilisateurs : les étudiants et les administrateurs. L'architecture repose sur une approche de microservices, orchestrée par une API Gateway, avec des bases de données spécialisées et des services interconnectés pour gérer les profils, les données académiques, les recommandations et l'intelligence artificielle. Ce rapport détaille chaque composant, leurs responsabilités, et les flux de données.

2 Composants de l'Architecture

2.1 Frontend

Le frontend est l'interface utilisateur accessible aux deux rôles : **étudiants** et **administrateurs**. Il permet aux utilisateurs d'interagir avec le système via des requêtes HTTP envoyées à l'API Gateway. Les fonctionnalités incluent la gestion des profils pour les étudiants et la modification des données académiques pour les administrateurs.

2.2 API Gateway

L'API Gateway agit comme un point d'entrée unique pour toutes les requêtes du frontend. Elle route les demandes vers les microservices appropriés en fonction de leur nature (authentification, gestion de données, etc.). Cela simplifie la communication et permet une gestion centralisée des autorisations et de la sécurité.

2.3 Microservices

L'architecture est composée de plusieurs microservices, chacun ayant une responsabilité spécifique :

1. **Service d'Authentification et de Gestion de Profils**
   - *Rôle* : Gère l'authentification des utilisateurs (étudiants et administrateurs) et la gestion de leurs profils.
   - *Base de données* : Utilise Neo4j pour stocker les données des utilisateurs, adapté aux relations complexes.
   - *Interaction* : Fournit des données au service de détection des changements lorsqu'un profil est modifié.
2. **Service de Gestion des Données Académiques**

— *Rôle* : Permet aux administrateurs de créer ou modifier les données académiques (cours, notes).
— *Base de données* : Stocke les données dans une instance séparée de Neo4j.
— *Interaction* : Envoie les modifications au service de détection des changements.

3. **Service de Détection des Changements**
   — *Rôle* : Surveille les modifications dans les profils ou données académiques.
   — *Interaction* : Notifie le service de réajustement via un mécanisme événementiel.
4. **Service de Réajustement**
   — *Rôle* : Orchestre les actions suite aux changements détectés (envoi au service d'IA ou de recommandation).
   — *Interaction* : Communique avec le service d'IA et le service de recommandation.
5. **Service de Recommandation**
   — *Rôle* : Génère des recommandations personnalisées pour les étudiants.
   — *Base de données* : Stocke les recommandations dans une base dédiée.
   — *Interaction* : Reçoit les profils modifiés et utilise les données du service d'IA.
6. **Service d'IA**
   — *Rôle* : Récupère des données externes et entraîne des modèles pour les recommandations.
   — *Interaction* : Reçoit des déclencheurs et fournit des données entraînées.

2.4 Bases de Données

— **Neo4j (Profils Utilisateurs)** : Stocke les informations des utilisateurs avec une structure en graphe.
— **Neo4j (Données Académiques)** : Contient les données académiques, séparées des profils.
— **Neo4j (Base de Recommandations)** : Stocke les recommandations générées.

3 Flux de Données

Exemple de flux :

1. Un administrateur modifie une donnée académique via le frontend.
2. La requête est envoyée à l'API Gateway, qui la route vers le *Service de Gestion des Données Académiques*.
3. Ce service met à jour la base Neo4j (Données Académiques).
4. Le *Service de Détection des Changements* détecte la modification et notifie le *Service de Réajustement*.
5. Le *Service de Réajustement* envoie les données au *Service d'IA* pour un réentraînement.
6. Le *Service d'IA* occupe l'entrainement du modèle et faire les prédictions.
7. Le *Service de Recommandation* envoie student profile comme input au service d'IA pour récupérer les diplômes recommandés.

4 Représentation Visuelle

Le diagramme suivant illustre l'architecture :

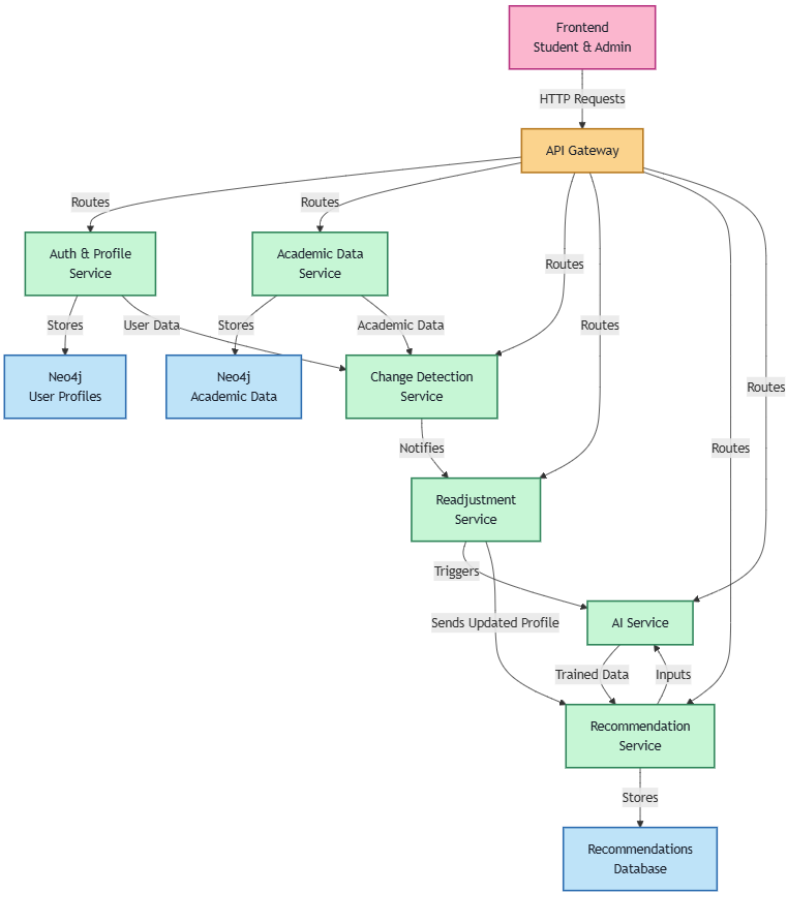

FIGURE 1 – Caption for the figure

5 Conclusion

Cette architecture est robuste et adaptée à un système avec deux types d'utilisateurs. Elle combine une gestion efficace des données relationnelles (via Neo4j) avec une approche modulaire et réactive. Pour une mise en œuvre optimale, préciser les technologies de communication et le type de base de données pour les recommandations serait utile.