App Neo4j

TD : Neo4j avec Flask

Objectifs

- Créer une API RESTful en Python avec Flask
- Configurer une base de données Neo4j
- Définir des modèles pour les utilisateurs, leurs relations, les posts et les commentaires
- Implémenter des routes CRUD pour gérer les utilisateurs, les posts et les commentaires

Prérequis

- Python 3.8+
- Docker
- Neo4j (via Docker ou installation locale)
- Postman ou curl pour tester l'API

Étapes

1. Initialisation du projet

- Créez un nouveau projet Python
- Installez les dépendances nécessaires :

```
pip install flask py2neo
```

- Créez un conteneur Docker pour la base de données Neo4j :

```
docker run --name neo4j -d \\
  -p 7474:7474 -p 7687:7687 \\
  -e NEO4J_AUTH=neo4j/password \\
  neo4j
```

2. Définition des modèles

Les modèles sont gérés avec `py2neo`.

- **Utilisateur**
  - `name` : Nom
  - `email` : Email
  - `created_at` : Date de création
- **Post**
  - `title` : Titre
  - `content` : Contenu
  - `created_at` : Date de création
- **Commentaire**
  - `content` : Contenu
  - `created_at` : Date de création

> Important : Un commentaire doit obligatoirement être lié à un post.

- **Relations**
  - `CREATED` : Indique qu'un utilisateur a créé un post ou un commentaire
  - `HAS_COMMENT` : Lie un post à un commentaire (chaque commentaire doit être rattaché à un post)
  - `FRIENDS_WITH` : Relation d'amitié entre utilisateurs
  - `LIKES` : Relation indiquant qu'un utilisateur a aimé un post ou un commentaire

3. Routes demandées

Routes pour les utilisateurs

- `GET /users` : Récupérer la liste des utilisateurs
- `POST /users` : Créer un nouvel utilisateur
- `GET /users/:id` : Récupérer un utilisateur par son ID
- `PUT /users/:id` : Mettre à jour un utilisateur par son ID
- `DELETE /users/:id` : Supprimer un utilisateur par son ID
- `GET /users/:id/friends` : Récupérer la liste des amis d'un utilisateur
- `POST /users/:id/friends` : Ajouter un ami (ID de l'ami dans le body)
- `DELETE /users/:id/friends/:friendId` : Supprimer un ami
- `GET /users/:id/friends/:friendId` : Vérifier si deux utilisateurs sont amis
- `GET /users/:id/mutual-friends/:otherId` : Récupérer les amis en commun

Routes pour les posts

- `GET /posts` : Récupérer tous les posts
- `GET /posts/:id` : Récupérer un post par son ID
- `GET /users/:id/posts` : Récupérer les posts d'un utilisateur
- `POST /users/:id/posts` : Créer un post

> Lors de la création, le post doit être lié à son créateur via une relation CREATED.

- `PUT /posts/:id` : Mettre à jour un post
- `DELETE /posts/:id` : Supprimer un post
- `POST /posts/:id/like` : Ajouter un like à un post (créer la relation `LIKES` entre un utilisateur et le post)

- `DELETE /posts/:id/like` : Retirer un like d'un post (supprimer la relation `LIKES`)

Routes pour les commentaires

- `GET /posts/:id/comments` : Récupérer les commentaires d'un post
- `POST /posts/:id/comments` : Ajouter un commentaire

> Lors de la création d'un commentaire, deux relations doivent être établies :
>
> - Une relation `CREATED` entre l'utilisateur et le commentaire
> - Une relation `HAS_COMMENT` entre le post et le commentaire

- `DELETE /posts/:postId/comments/:commentId` : Supprimer un commentaire
- `GET /comments` : Récupérer tous les commentaires
- `GET /comments/:id` : Récupérer un commentaire par son ID
- `PUT /comments/:id` : Mettre à jour un commentaire
- `DELETE /comments/:id` : Supprimer un commentaire
- `POST /comments/:id/like` : Ajouter un like à un commentaire (créer la relation `LIKES` entre un utilisateur et le commentaire)
- `DELETE /comments/:id/like` : Retirer un like d'un commentaire (supprimer la relation `LIKES`)

4. Exemples de requêtes Cypher

- **Créer un utilisateur :**

```
CREATE (u:User {name: 'Alice', email: 'alice@example.com', created_at: timestamp()})
RETURN u
```

- **Créer un post et lier le créateur :**

```
MATCH (u:User {id: 'user123'})
CREATE (p:Post {title: 'Mon premier post', content: 'Contenu du post', created_at: timestamp()})
CREATE (u)-[:CREATED]→(p)
RETURN p
```

- **Créer un commentaire, lier le créateur et le post (obligatoire) :**

```
MATCH (u:User {id: 'user123'}), (p:Post {id: 'post123'})
CREATE (c:Comment {content: 'Super post !', created_at: timestamp()})
CREATE (u)-[:CREATED]→(c)
CREATE (p)-[:HAS_COMMENT]→(c)
RETURN c
```

- **Ajouter un like à un post :**

```
MATCH (u:User {id: 'user123'}), (p:Post {id: 'post123'})
CREATE (u)-[:LIKES]→(p)
RETURN p
```

- **Supprimer un like d'un post :**

```
MATCH (u:User {id: 'user123'})-[l:LIKES]→(p:Post {id: 'post123'})
DELETE l
RETURN p
```

Ressources utiles

- Flask
- py2neo
- Neo4j
- Neo4j Docker

Livrables

- Lien vers le repository GitHub du projet
- Documentation expliquant l'installation et l'exécution du projet
- Guide de test des routes de l'API (exemples Postman ou `curl`)

Critères d'évaluation

- Respect des consignes
- Qualité du code
- Documentation
- Fonctionnalités implémentées
- Gestion des erreurs