IDV-WEB4 / API REST / Etape 4 (Java)

Modalités

Type	Description
Dépôt	https://rendu-git.etna-alternance.net/module-9438/activity-51077/group-1010680
Dossier	Etape_4/
Correction	Moulinette / Correction à distance
Taille de groupe	Groupe de 2

Objectifs

Notion	Description
SpringBoot	Ajouter de nouvelles entités dans SpringBoot et implémenter des tests unitaires

Consignes

Avant de commencer, vous devez copier votre projet de l'étape 3 dans un dossier nommé `Etape_4`.

1) Ajout d'une entité `Address`

Vous devez ajouter une nouvelle entité `Address` avec les paramètres suivants :

- `id` : auto-généré et unique
- `street` : chaîne de caractères (non vide, maximum 100 caractères)
- `postalCode` : chaîne de caractères (non vide, maximum 30 caractères)
- `city` : chaîne de caractères (non vide, maximum 50 caractères)
- `country` : chaîne de caractères (non vide, maximum 50 caractères)
- `user` : relation *Many-To-One*
- `creationDate` : date
- `updatedDate` : date

La date de création et modification doivent être définies au moment de la création.
La date de modification doit être modifiée à chaque modification de l'entité.

Concernant la relation *Many-To-One*, un utilisateur peut avoir plusieurs adresses, mais une adresse ne peut avoir qu'un seul utilisateur.

Vous devez également créer son repository avec les méthodes dont vous aurez besoin.

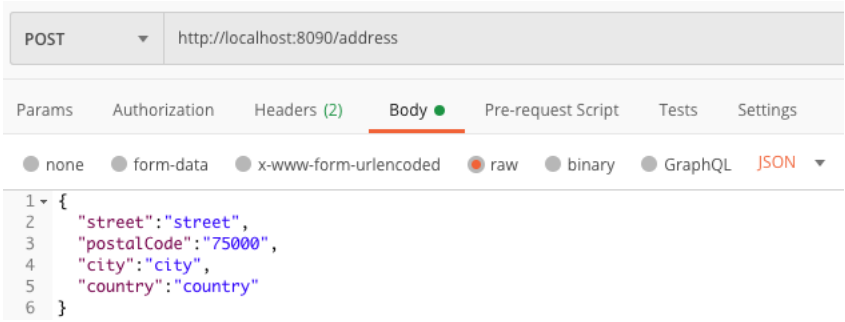
Voici le describe de la table MySQL à respecter :

```
mysql> DESCRIBE address;
+---------------+--------------+------+-----+---------+----------
| Field         | Type         | Null | Key | Default | Extra
+---------------+--------------+------+-----+---------+----------
| id            | int          | NO   | PRI | NULL    | auto_incr
| user_id       | int          | NO   | MUL | NULL    |
| street        | varchar(100) | NO   |     | NULL    |
| postal_code   | varchar(30)  | NO   |     | NULL    |
| city          | varchar(50)  | NO   |     | NULL    |
| country       | varchar(50)  | NO   |     | NULL    |
| creation_date | datetime     | YES  |     | NULL    |
| updated_date  | datetime     | YES  |     | NULL    |
+---------------+--------------+------+-----+---------+----------
8 rows in set (0,00 sec)
```

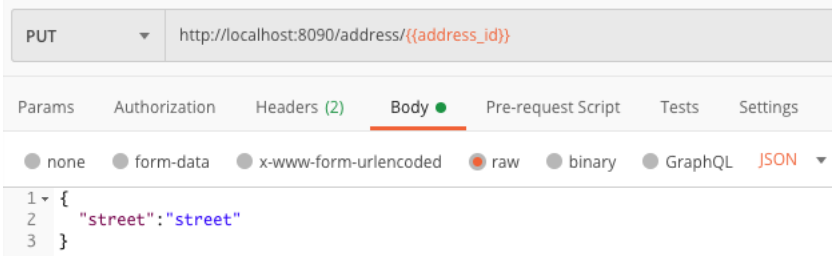
2) Création de son contrôleur REST

Vous devez créer un contrôleur REST nommé `AddressController` préfixé en `/address` qui contiendra les méthodes suivantes :

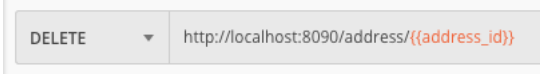
- **GET** : pour récupérer une liste ou une seule adresse
  Exemples:
  - http://localhost:8090/address/ renvoie la liste des adresse
  - http://localhost:8090/address/1/ ne renvoie que l'adresse d'ID #1.
- **POST** : création d'une nouvelle adresse
  - Retourne l'entité “address” récemment créé



- **PUT** : Modification d'un adresse
  - Retourne l'entité “address” mise à jour



- **DELETE** : Suppression d'une adresse
  - Retourne un JSON avec une clé “success” à TRUE ou FALSE



Attention:

Tous les appels API doivent répondre en JSON avec le bon code HTTP.
Un utilisateur avec le rôle `USER` ne peut pas accéder aux adresses qui ne lui appartiennent pas. Un utilisateur avec le rôle `ADMIN` a tout le contrôle.
La propriété d'une adresse est basée sur l'utilisateur qui en demande la création.
Lors d'un update, tous les paramètres n'ont pas besoin d'être envoyés, seulement ceux qui ont besoin d'être mis à jour.

3) Ajout d'un contrôleur REST pour les utilisateurs

Vous devez créer un contrôleur REST nommé `UserController` préfixé en `/user`, qui contiendra les méthodes suivantes :

- **GET** : Pour récupérer la liste des utilisateurs, ou d'un utilisateur précis.
- **PUT** : Pour modifier le rôle et le username de l'utilisateur.
- **DELETE** : Pour supprimer un utilisateur.

Les appels doivent suivre la même logique que pour les adresses.

Attention :

Il faut obligatoirement être connecté pour communiquer avec ces routes.
Un utilisateur avec le **ROLE_USER** ne pourra modifier ou supprimer que le username de son propre profil mais pourra consulter les autres utilisateurs.
Un utilisateur avec le **ROLE_ADMIN** aura le contrôle complet sur tous les utilisateurs.

4) Tests unitaires

Vous allez créer une première classe de tests unitaires appelée `ControllerTests` qui utilisera `MockMvc`.

Dans une première fonction nommée `testAuthenticate` vous devez tester que :

- la route `/register` répond bien en **201**.
- si vous rappelez `/register` avec les mêmes paramètres, vous obtenez bien une réponse **409** car l'utilisateur existe déjà.
- la route `/authenticate` retourne bien un statut **200** ainsi que votre token.
- la route `/me` retourne un statut **200** avec les informations de l'utilisateur (attention à bien penser au token Bearer).

Dans une seconde fonction nommée `testUser` vous devez tester que :

- sans token Bearer, la route `/user` retourne bien un statut **401**
- avec un token Bearer valide, la route `/user` retourne bien un statut **200**.
- avec un **ROLE_USER**, la suppression retourne bien un statut **403**.
- avec un **ROLE_ADMIN**, la suppression retourne bien un statut **200**.

Dans une dernière fonction nommée `testAddress`, vous devez tester que :

- sans token Bearer, la route `/address/` retourne bien un statut **401**.
- avec un token Bearer valide, la route `/address/` retourne bien un statut **200**.
- avec un token Bearer valide, l'ajout d'une adresse retourne bien un statut **201**.
- avec un **ROLE_USER**, la suppression d'une adresse qui n'est pas la sienne retourne bien un statut **403**.
- avec un **ROLE_ADMIN**, la suppression d'une adresse qui n'est pas la sienne retourne bien un statut **200**.

Attention :

Avant de faire une demande de validation, assurez-vous que tous vos tests fonctionnent et que tous les fichiers nécessaires à leur fonctionnement sont bien commités.

Remarques

Les tests unitaires doivent être créés dans le bon dossier (`src/test/java`).