Framework

Développement d'une API REST avec Spring-boot

Exercice pour le Vendredi 13 décembre 2019

Soyez curieux, n'hésitez pas à vous renseigner sur spring-boot, spring, spring-data-jpa sur internet. Mais essayez de garder au maximum votre code simple. Basez-vous sur le projet que je vous donne, analysez-le, rappeler vous comment on fait une injection de dépendance, comment on déclare un service, un controller, …

Je serai dispo sur discord sur le serveur iut-paris8. Rejoignez #csid (j'essaierai de répondre au maximum). Invitation https://discord.gg/SXP9qxB
Je suis aussi disponible sur IRC sur les serveurs Quakenet, sur #csid

1. Cloner le projet https://github.com/Kaway/csid-enterprise
Ce projet est ma version de l'exercice que nous avons fait en cours jusque là.

2. Modifier le fichier *resources/application.properties* pour que la configuration de la base de données soit correcte (vous pouvez utiliser une base de données h2, dite *in-memory*, au lieu de PostgreSQL pour MySQL pour que ce soit plus simple)

```
#DATABASE
spring.datasource.url=jdbc:h2:mem:csid
spring.datasource.username=csid
spring.datasource.password=csid
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database=mysql

spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
```

(Vous trouverez le projet déjà configuré sur la branche "h2_config" dans le repo github)

3. Lancer le projet (avec Spring-boot, il suffit d'exécuter la classe contenant notre ***main*** méthode.

4. Avec votre collection Postman, vérifier que le projet est correct:
- Récupérer la liste de tous les utilisateurs
- Créer un utilisateur
- Récupérer à nouveau la liste de tous les utilisateurs
- Récupérer l'utilisateur créé
- Ajouter n utilisateurs
- Supprimer un utilisateur

- Modifier un utilisateur avec PUT
- Modifier un utilisateur avec PATCH (ne modifier qu'un seul champ firstName ou lastName)

5. Ajouter une Address à notre Employee:
   - dans le DTO, l'adresse sera de la forme addressLine1 et addressLine2 (string)
   - dans le model, l'adresse sera un objet (numero, nom rue, code postal, ville, pays)
   - dans l'Entity, l'adresse ne sera pas un objet, les champs appartiendront à l'objet Employee

Enregistrer l'adresse en base et faire en sorte qu'elle soit disponible dans la réponse de notre API.

Cet exemple vous montre pourquoi un découpage DTO, Model, Entity est intéressant dans certaines applications

6. On souhaite déclencher le versement de la paie pour nos employés. Notre système n'étant qu'à ses débuts, on doit effectuer l'opération employé par employé. Chaque fois que la paie d'un employé est versé on enregistre le salaire versé pour le mois.
   - Ajouter un champ ***salary*** à notre employé: ce champ représente le salaire mensuel d'un employé pour un mois complet (21 jours)
   - Faire un service, controller, repository permettant de récupérer l'ensemble des salaires versés.

Un salaire est un objet constitué d'un employé, d'un montant versé, du mois/année du salaire, de la date à laquelle on a déclenché le versement et du nombre de jour travaillés
La modélisation du/des différent(s) objet(s) (DTO, Entity et Model) est de votre ressort.

Pour calculer le salaire à verser, on prendra en paramètre (en plus des autres que vous devez trouver vous-mêmes) le nombre de jours travaillés par notre employé (une règle de trois)

Contraintes:
- on ne peut pas verser 2 fois le salaire d'un employé pour le même mois.
- on ne peut pas modifier un salaire déjà versé
- un employé travaille entre 0 et 21 jours par mois

7. Récupérer la liste des salaires versés pour un salarié donné (***/employee/{id}/salary***)

Bonus: on souhaite obtenir la liste des salaires par date de versement croissante et décroissante.
Bonus 2: on souhaite aussi obtenir la liste des salaire par mois de salaire (selon votre modélisation, ça peut changer des choses ou non)

Ressources utiles:
- @ManyToOne, @OneToOne, @OneToMany pour les jointures
- @RequestParam pour le tri