Projet Spring-boot pour le 14/01/2019

Vos amis :

- https://spring.io/projects/spring-boot
- généralement google est votre ami ...

Pour commencez clonez ce projet :
https://github.com/marwensaid/training-spring-boot.git

Attention : Vous devez bien avoir forké le projet avant de le cloner. Cette opération copie le projet dans votre compte GitHub et vous permet de pousser les commits résultant de votre travail.
Puis implémentez les 2 méthodes suivantes dans ProductController.java :

1. calculerMargeProduit , qui calcule la marge de chaque produit (différence entre prix d'achat et prix de vente).
2. trierProduitsParOrdreAlphabetique qui retournera la liste de tous les produits triés par nom croissant

Ensuite, ajoutez à la méthode ajouterProduit une condition afin de vérifier que le prix de vente n'est pas égal à 0.
Enfin, testez les 2 méthodes crées avec *Postman*

Consignes pour la réalisation

A la fin de chaque partie, créez un *commit* et poussez le vers votre projet *GitHub*

Partie 1 - Affichage de la marge

La méthode calculerMargeProduit doit répondre à une requête *GET* sur l'URI /AdminProduits. Les données doivent être récupérées depuis la base de données mises en place dans le projet.
Voici un exemple de réponse attendue :

```
{
"Product{id=1, nom='Ordinateur portable', prix=350}": 230,
"Product{id=2, nom='Aspirateur Robot', prix=500}": 300,
"Product{id=3, nom='Table de Ping Pong', prix=750}": 350
}
```

Partie 2 - Tri par ordre alphabétique

La méthode trierProduitsParOrdreAlphabetique doit impérativement faire appel à une méthode que vous allez ajouter dans ProductDao qui utilise le nommage conventionné de

Spring Data JPA pour générer automatiquement les requêtes. Voici le résultat à obtenir avec le contenu de la base de données du cours :

{
{
"id": 2,
"nom": "Aspirateur Robot",
"prix": 500,
"prixAchat": 200
},
{
"id": 1,
"nom": "Ordinateur portable",
"prix": 350,
"prixAchat": 120
},
{
"id": 3,
"nom": "Table de Ping Pong",
"prix": 750,
"prixAchat": 400
}
}

Partie 3 - Validation du prix de vente

Si le prix de vente est de 0, lancez une exception du nom de ProduitGratuitException (à créer) qui retournera *le bon code HTTP* pour ce cas avec un message explicatif que vous définirez.

Partie 4 - Mettre en place des tests unitaires

1. Mettre en place des tests unitaires (réels) pour votre code
2. votre couverture de testes doit être au moins à 80%
3. L'utilisation du TDD est recommandé (un petit mot sur la méthodologie pendant la soutenance)

Partie 5 - Test des méthodes créées

Suivez les étapes suivantes pour écrire la collection, l'exécuter et fournir le résultat pour votre testeur :

4. Créez un dossier TestsPostman à la racine de votre projet
5. Dans Postman, créez une collection pour définir vos tests
6. Exportez la collection au format *JSON* dans le dossier TestsPostman
7. Commitez et poussez vers le dépôt *GitHub*

Partie 6 - Swagger

Documentez votre microservice avec Swagger 2

Bonus : Mettez un font à votre application spring-boot

à vous de jouer … cette partie est free … vous pouvez utiliser ce que vous voulez pour le front qui marche avec spring-boot

Livrables

1. le lien de votre projet github
2. la collection postman dans laquelle il y a vos requête de test

Une mini soutenance est planifié le 14/01/2019 pour présenter votre code et vos testes (10 min max par projet)

Votre dépôt doit contenir les *commits* pour chaque partie.