# Projet Microservices / Spring M2 DFS

Vos amis :

- Tous les TD que nous avons réalisé
- 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 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.*

# Consignes pour la réalisation

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

## Partie 0 - Implémentation des endpoint : delete / update & GetById

l'implémentation de ces méthodes c'est dans le Controller

## Partie 1 - Affichage de la marge

La méthode calculerMargeProduit (différence entre prix d'achat et prix de vente)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 (qui retournera la liste de tous les produits triés par nom croissant )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 :


```
{
{
"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 allez définir.

# Partie 4 - Circuit Breaker

Mettez en place un Circuit Breaker pour chaque endpoint pour faire en sort d'afficher un message d'erreur, un snapshot des données, ou une methode de fallback  quand le microservice microcommerce est indisponible (je pense vous avez compris qu'il faut monter un autre microservices spring boot pour faire ce circuit Breaker vu que l'appel est de l'extérieur)

## Partie 5 - 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 60%
3. L'utilisation du TDD est recommandé (aucune obligation)

## Partie 6 -Microservice Eureka Server

Créez un microservice qui portera l'Eureka serveur.

Les deux microservices "microcommerce" et "myMicroservice" vont s'enregistrer dans le microservice "Eureka Server"

## Partie 7 - Spring Security (Bonus n°1)

Mettez en place une authentification lors de l'affichage de produit avec deux niveau de role (user et admin)

y'a que l'admin qui peut voir les prix des produit

## Partie 8 - Gestion des erreurs HTTP

en utilisant la lib HTTP et spring gérer les cas 4xx et 5xx et faites des TU pour simuler un cas de 400 et 500

## Partie 9 - Swagger

Documentez vos microservice avec Swagger 2

## *Bonus pour tous le monde : 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 / ou le zip

Votre dépôt doit contenir les *commits* pour chaque partie ainsi que de chaque contributeur du projet (ajoutez moi dans votre repo github)