# TP1P2 – SpringBoot avec BD, callback et Rest API

## Consignes

* À faire seul.
* Compte pour 10% de la note.
* À remettre selon la date de remise inscrite dans Léa.

## Partie 1

Vous devez développer une application backend avec Spring Boot pour gérer un système de tickets pour du support technique.

Chaque ticket contient :

* un titre
* une description
* un statut (NOUVEAU, EN\_COURS, FERME) 🡪Enum
* une date de création
* une date de fermeture (nullable)
* une durée ouverte (en jours), champ calculé 🡪 n’est pas sauvegardé en BD

## À FAIRE :

1. Créer un nouveau projet **SpringBoot** et ajouter la dépendance à la base de données **H2**.
2. Créer l’entité **Ticket** avec les champs mentionnés.
3. Ajouter les annotations JPA nécessaires pour que l’interaction avec la BD fonctionne.
4. Avant de continuer, vous aurez besoin de connaître et comprendre le fonctionnement de certaines annotations

Expliquez le rôle des annotations suivantes :

|  |
| --- |
| @PrePersist : S’execute avant qu’un nouvel objet soit enregistre dans la base de donnee et permet d’initialiser des champ automatiquement |
|  |
|  |
|  |
|  |

|  |
| --- |
| @PreUpdate: |
| S’excute avant qu’un objet existant soit mis a jour et permet de mettre a jour certaine |
| Valeurs automatiquement |
|  |
|  |

|  |
| --- |
| @PostLoad: |
| S’execute juste après qu’un objet est charge depuis la bd et permet d’effectuer |
| Des actions après la recuperation |
|  |
|  |

|  |
| --- |
| @PreRemove: |
| S’execute juste avant qu’un objet soit supprimer de la bd et permet d’effectuer des vérifications ous auvegarde |
|  |
|  |
|  |
|  |

|  |
| --- |
| @ EntityListeners: |
| Permet de lier une classe externe |
|  |
|  |
|  |
|  |

|  |
| --- |
| @ Service: |
| Indique qu’une classe est un service dans Spring |
|  |
|  |
|  |
|  |

1. Implémenter une classe **TicketListener** qui contiendra les callbacks JPA :
   * @PrePersist : fixer automatiquement la date de création avant insertion en base. Afficher le message « CALLBACK @PrePersist: dateCréation fixée à pour le nouveau ticket ».
   * @PreUpdate : si le statut change à FERME et que la date de fermeture est nulle, fixer la date de fermeture. Afficher le message « Callback @PreUpdate : dateFermeture fixée à pour ticket id=1 ».
   * @PostLoad : calculer la durée ouverte en jours (différence entre date création et date fermeture ou aujourd’hui) et la stocker dans un champ transitoire (@Transient). Afficher le message « Callback @PostLoad : dureeOuverte calculée à 2 jours pour ticket id=1 ».
   * @PreRemove : afficher un message dans la console lors de la suppression du ticket. Afficher le message « Callback @PreRemove : suppression du ticket id=1 avec le titre=Problème imprimante ».
2. Attacher ce listener à l’entité avec @EntityListeners.
3. Tester le fonctionnement avec CommandLineRunner
   * Créer **trois** ticket,
   * Modifier la date de création du **ticket #2** ( avec SetDateCreation()) pour mettre 1 semaine avant aujourd’hui.
   * Modifier le statut du **ticket #2** pour « fermer ».
   * Récupérer et afficher les données pour vérifier la durée du **ticket #2**.
   * Supprimer un le **ticket #1**.
4. Ajouter un REST API qui permettra de voir tous les tickets présents en BD (vous devriez voir les tickets #2 et #3)
   * Créer une classe TicketService (@Service). Ajouter un @Autowired sur le ticketRepository et créer une méthode qui retourne une liste de tous les ticket en BD.
   * Créer une classe TicketController (@RestController). Ajouter un @Autowired sur le ticketService et créer une méthode (@GetMappping) qui retourne une liste de tous les tickets.
5. Testez le bon fonctionnement en ouvrant votre navigateur Internet à l’adresse de votre REST API.

Exemple :  
Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

## Partie 2 – CallBack et Lambda

Vous devez utiliser au moins une lambda dans vos exemples.

### Affichage d’une liste

Créer une interface ***ItemCallback*** avec une méthode ***affiche(String item)***.

Faire une classe AfficheListe qui ne prend aucun constructeur et qui possède qu’une seule méthode ***afficherItem(List<String> liste, ItemCallBack)***.

Dans la classe Main, continuer les instructions suivantes :

**List<String> items = List.*of*("Item1", "Item2", "Item3", "Item4");**

### Résultat

Crée une interface ***ResultCallback*** avec une méthode ***onResult(int result)***.

Faire classe sans constructeur et qui contient une méthode pour additionner 2 nombres. Cette méthode doit recevoir 2 nombres et le callback.

La classe Main doit utiliser la classe créée précédemment et faire afficher le résultat via le callback.