@PostMapping et @RequestBody

# Démonstration 4 du module 4

|  |
| --- |
| Les objectifs de cette démonstration :   * Réaliser un POST avec injection d’un objet JSON dans le body * Gérer les exceptions au travers d’un statut et d’un message |

Contexte

* Nous continuons notre application représentant une partie du cœur de métier de l’ENI Ecole.

Déroulement

# Sauver un nouvel employé

## Dans EmployeController

* Ajout d’un PostMapping avec un objet Employe en paramètre
  + Utilisation des annotations @PostMapping pour gérer la demande de POST et @RequestBody pour récupérer l’objet JSON du body
* Gestion des exceptions de la couche BLL et DAL avec un retour de statut

@PostMapping

**public** ResponseEntity<?> ajoutEmploye(@RequestBody Employe employe) {

**try** {

employeService.ajouter(employe);

**return** ResponseEntity.*ok*(employe);

} **catch** (RuntimeException e) {

// Erreur BLL ou DAL

**return** ResponseEntity.*status*(HttpStatus.***NOT\_ACCEPTABLE***).body(e.getMessage());

}

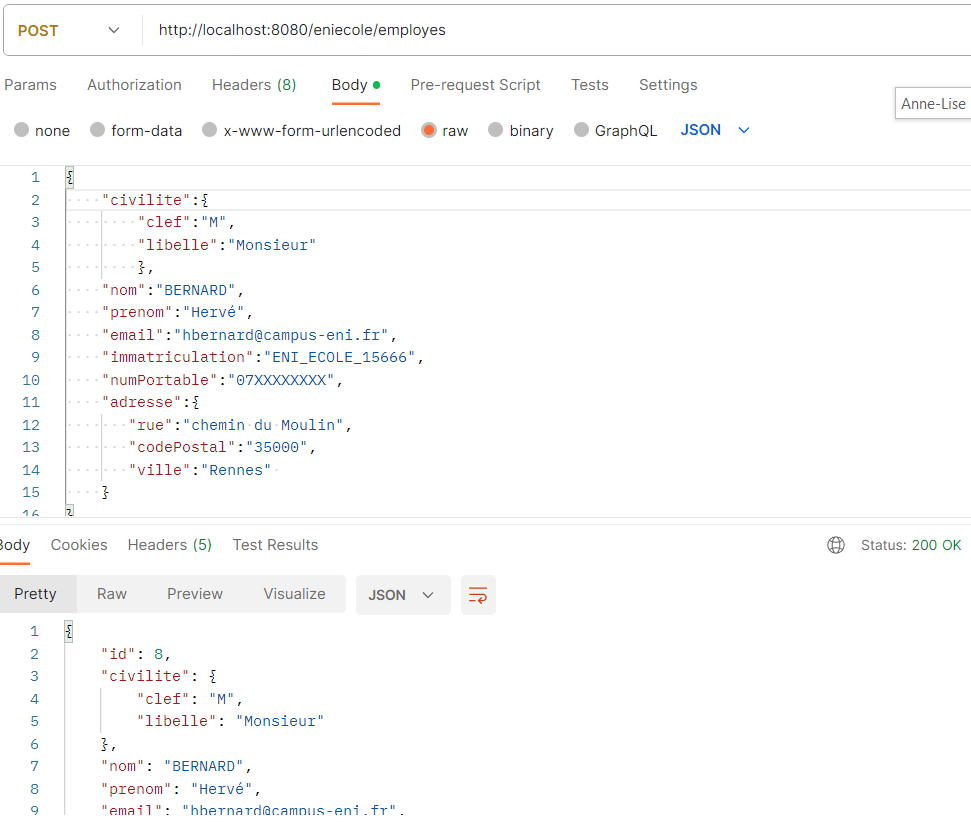
}

# Exécution avec Postman

* Dans Postman, il faut changer la méthode du protocole
  + Sélectionner POST
* Il faut ajouter l’objet JSON dans le body de la requête
  + Sélectionner body 🡪 raw 🡪 JSON

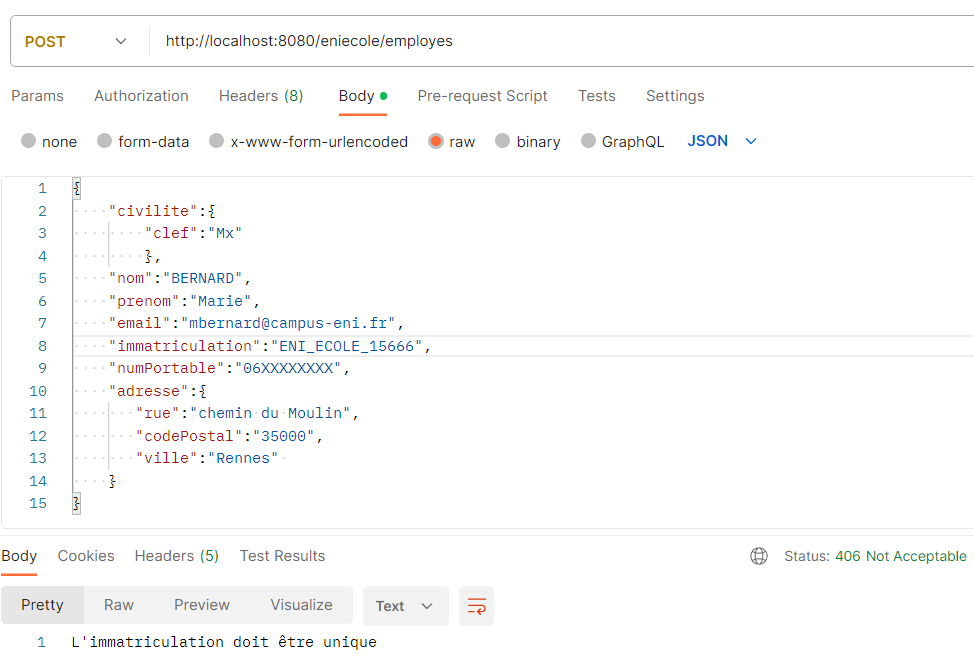
## Un Employe au format JSON complet

* URL : http://localhost:8080/eniecole/employes

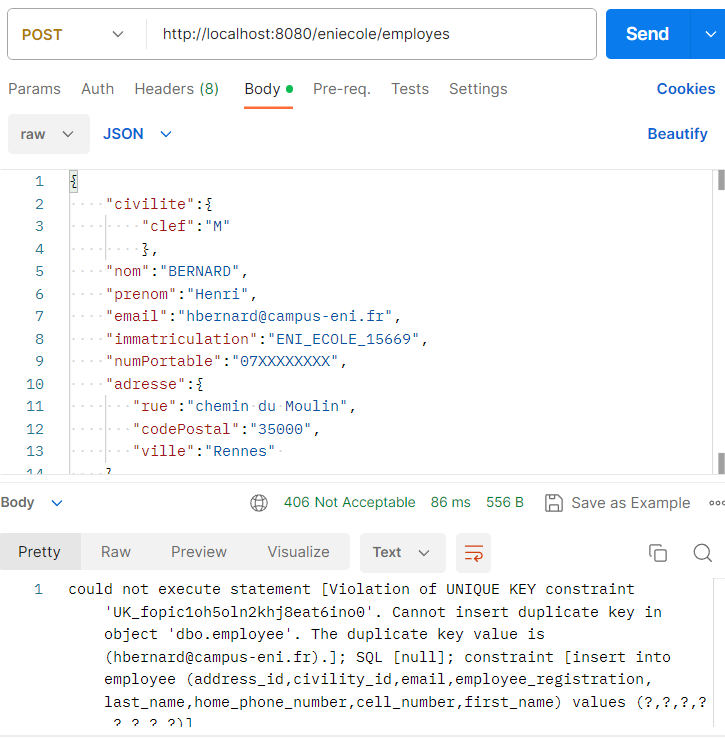


* Vous voyez les données en JSON dans la requête et dans la réponse
* Le statut de la réponse (200)

## Un Employe avec une immatriculation identique à un autre



* Vous voyez les données en JSON dans la requête
* Le statut de la réponse (406) 🡪 retour sur la levée d’exception de la couche BLL
* Le message provient de la BLL 🡪 précise que l’immatriculation doit être uniqueUn Employe avec une mail identique à un autre



* Vous voyez les données en JSON dans la requête
* Le statut de la réponse (406) 🡪 retour sur la levée d’exception de la couche DAL
* Le message provient de la DAL 🡪 il précise que la base interdit le duplicat sur email
  + Il serait plus intéressant de voir comment bloquer cela au niveau BLL et avec un message de l’application et pas de la base de données
  + Cela donne trop d’information sur la structure de la base (il faut protéger la ressource)

### Ajout d’une validation sur l’unicité de l’email

* Dans la classe EmployeService, il nous faut capturer les exceptions de la couche DAL
  + Et de transmettre un message d’erreur en conséquence
  + Les exceptions de la couche DAL sont des RuntimeException
  + Il est préférable de gérer par défaut toute erreur sous cette classe avec un message générique

@Override

**public** **void** ajouter(Employe employe) {

// Validation des données de l'employé avant sauvegarde

**if** (employe == **null**) {

**throw** **new** RuntimeException("L'employé n'est pas renseigné");

}

validerImmatriculation(employe);

validerChaineNonNulle(employe.getNom(), "Vous devez renseigner le nom");

validerChaineNonNulle(employe.getPrenom(), "Vous devez renseigner le prénom");

validerChaineNonNulle(employe.getEmail(), "Vous devez renseigner un email");

**try** {

employeRepository.save(employe);

} **catch** (RuntimeException e) {

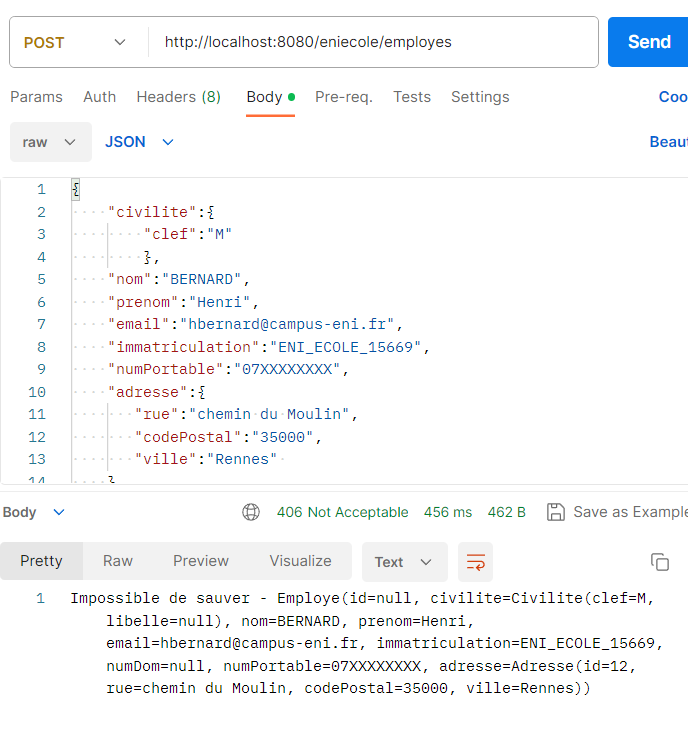
**throw** **new** RuntimeException("Impossible de sauver - " + employe.toString());

}

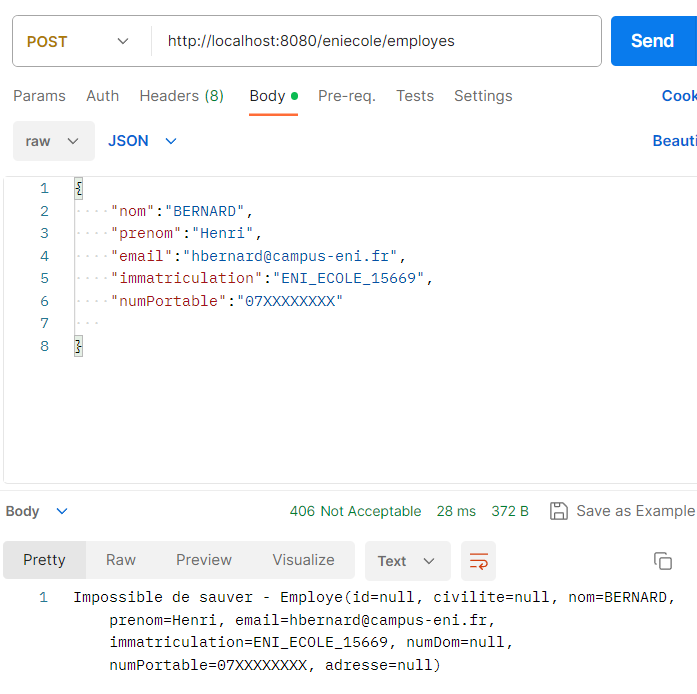
}

### Relancer l’exécution

* Maintenant, nous avons la capture de la RuntimeException de la couche DAL
  + Et le message de la BLL



## Un Employe sans Adresse et sans Civilite



* Avec le try-catch autour de la méthode save dans la BLL, nous avons bien une capture de l’exception de la couche DAL
  + Mais il serait intéressant d’empêcher d’aller jusqu’à la base pour se rendre compte qu’il n’y a pas de Civilite et d’Adresse

Nous allons voir par la suite comment placer des validations sur les BO et les activer dans le contrôleur pour ne pas aller jusqu’à la base