@GetMapping

# Démonstration 2 du module 4

|  |
| --- |
| Les objectifs de cette démonstration :   * L’ajout du starter Spring Web * La création d’un contrôleur REST avec l’annotation @RestController * La gestion d’appels GET depuis le client avec @GetMapping * Des données retournées de types simples String * Des données retournées de type POJO * Utilisation de ResponseEntity pour transmettre un statut et un POJO * Utilisation de Postman pour manipuler ces premières web service |

Contexte

* Nous reprenons notre application représentant une partie du cœur de métier de l’ENI Ecole.
  + Il s’agit de l’application demo-eni-ecole
  + Si votre application n’est pas complète, vous pouvez reprendre le code Java dans les ressources
  + Attention, le paramétrage du projet Spring Boot n’est pas inclus
    - Il vous faut reprendre les démonstrations des modules 1 et 2 pour valider les starters de Spring Boot
* Dans cette itération, nous voulons faire évoluer l’application vers une application Web Service REST

Déroulement

# Ajout du starter Spring Web

* Ouvrir build.gradle :
  + Ajout du starter Spring Web

//Ajout du starter Spring Web

**implementation** 'org.springframework.boot:spring-boot-starter-web'

* Puis, clic droit sur le projet
  + Gradle 🡪 Refresh Gradle Project
* Le projet est devenu un projet Web
  + Vous pouvez voir dans les dépendances
    - spring-boot-starter-web-3.X.X.jar
    - spring-web-6.X.X.jar
    - jackson-…jar
    - …
  + Ces librairies permettent d’accéder au contexte Web et REST avec Spring Boot

# Ajout du contrôleur EniController

Ce contrôleur est là pour présenter le transfert d’information simple : String

* Il va nous permettre d’accueillir les utilisateurs sur l’application
* Déclarer l’URL racine de l’API Eni Ecole
* Et retourner un message de bienvenue
* Création de la classe et poser l’annotation @RestController
* Pour préciser l’URL de l’API par défaut placer : @RequestMapping("/eniecole")
* Ajout d’une méthode annotée @GetMapping qui retourne un String

**package** fr.eni.demo.controller;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/eniecole")

**public** **class** EniController {

@GetMapping

**public** String welcomeAPI() {

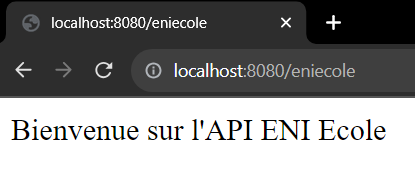
**return** "Bienvenue sur l'API ENI Ecole";

}

}

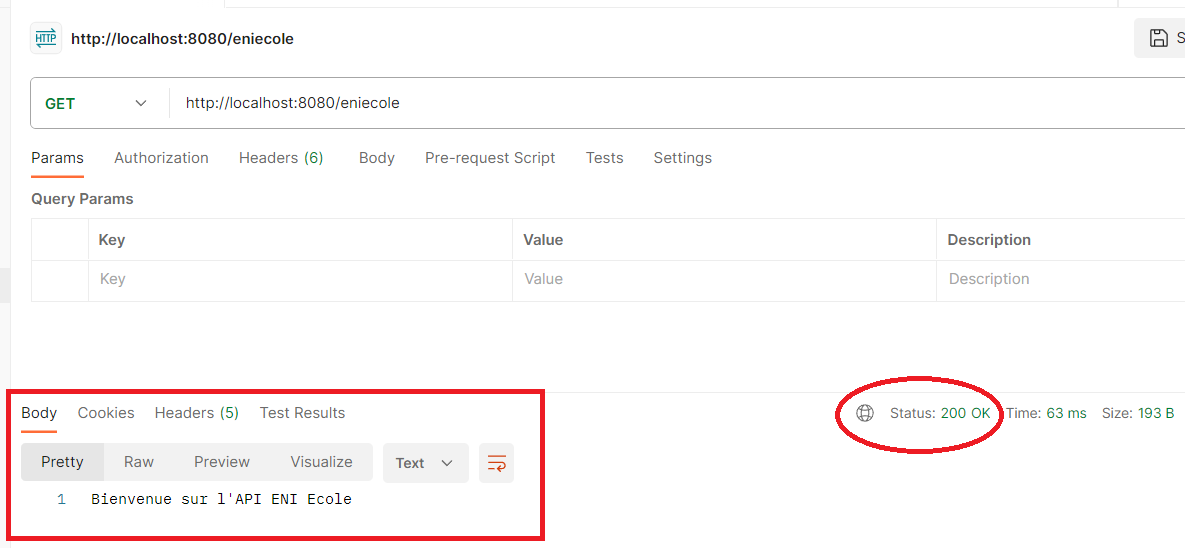
## Exécution dans un navigateur

* + Ouvrez un navigateur
  + Saisir l’URL : http://localhost:8080/eniecole
  + Vous obtiendrez, l’exécution de votre contrôleur et l’affichage de la chaîne de caractères :



## Exécution avec Postman

* Lancer Postman
* Cliquer pour créer une nouvelle requête
* Indiquer l’URL http://localhost:8080/eniecole
* Exécuter la requête



* Dans Postman, sans ouvrir d’outil de développement
  + Il est possible de voir le retour, le statut
  + Les informations sont facilement accessibles et lisibles
  + Et s’est rapide pour les manipulations

# Get – Renvoyer la liste des employés

## Données en base

* Il nous faut des données en base pour les utiliser pour les tests
* Nous vous fournissons une classe de tests dans les ressources : TestDatas.java
  + Copier la dans le package de test : fr.eni.demo.controller
  + Elle crée des civilités et des employés avec leur adresse
* Modifier la configuration de la base pour conserver les données
  + Dans application.properties :
  + Remplacer le create en update

spring.jpa.hibernate.ddl-auto=update

* Exécuter la classe de tests ; vous aurez donc des données en base
  + Vous pouvez les vérifier directement en base

select \* from civility;

select \* from address;

select e.\*, t.computer\_science\_course, b.office\_phone\_number from employee e

left outer join trainer t on e.employee\_id = t.employee\_id

left outer join school\_business\_officer b on e.employee\_id = b.employee\_id;

## EmployeController

* Ajout du contrôleur EmployeController
  + A cette itération, il va nous permettre de déclarer l’URL racine de l’API associée aux employés
  + Et la demande de la liste des employés au travers d’un GET
* Création de la classe et posée l’annotation @RestController
* Pour préciser l’URL de l’API par défaut placer : @RequestMapping("/api/employes")
* Nous avons déjà une couche BO, BLL et DAL associée à ce comportement.
  + Il faut injecter le service EmployeService dans le contrôleur
  + Utiliser l’annotation @AllArgsConstructor de Lombok pour gérer la création du constructeur avec le paramètre EmployeService
* Ajout d’un GetMapping
  + Création d’une méthode pour retourner l’ensemble des employés
  + Annotée @GetMapping
  + Pour le moment, utilisation de la librairie Jackson pour automatiquement transmettre la donnée en JSON
    - Rien à coder pour cela, la librairie fait tout le travail avec Spring

**package** fr.eni.demo.controller;

**import** java.util.List;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** fr.eni.demo.bll.EmployeService;

**import** fr.eni.demo.bo.Employe;

**import** lombok.AllArgsConstructor;

@AllArgsConstructor

@RestController

@RequestMapping("/eniecole/employes")

**public** **class** EmployeController {

**private** EmployeService employeService;

@GetMapping

**public** List<Employe> rechercherTousEmployes() {

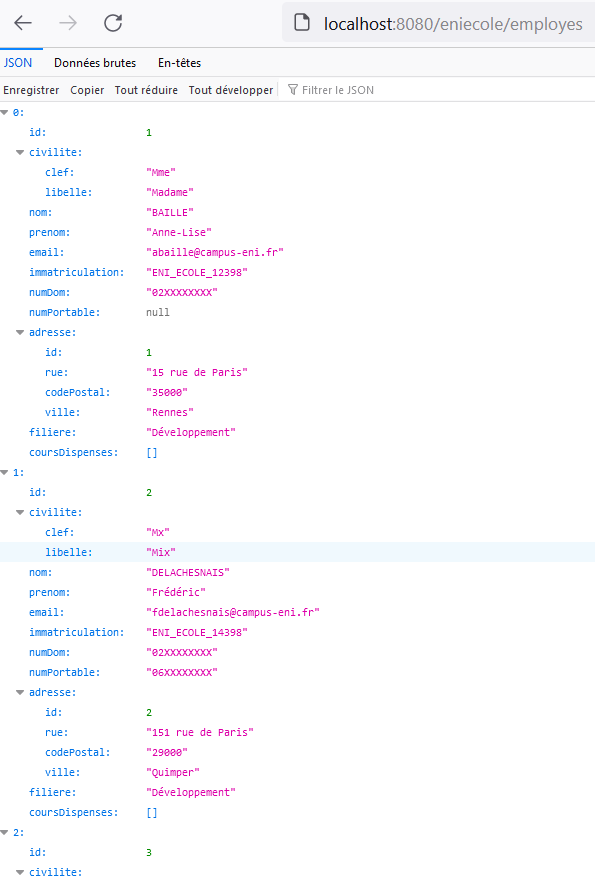
**return** employeService.chargerTousEmployes();

}

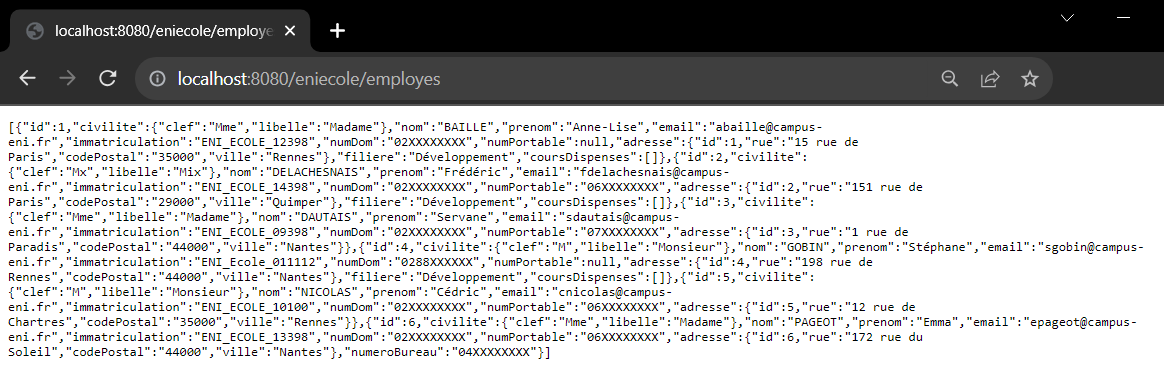
}

## Exécution dans un navigateur

* + Ouvrez un navigateur et saisir l’URL : http://localhost:8080/eniecole/employes
  + Vous obtiendrez, l’exécution de votre contrôleur et l’affichage du tableau des employés :
    - Sous Firefox, il nous affiche un rendu pour un tableau de JSON



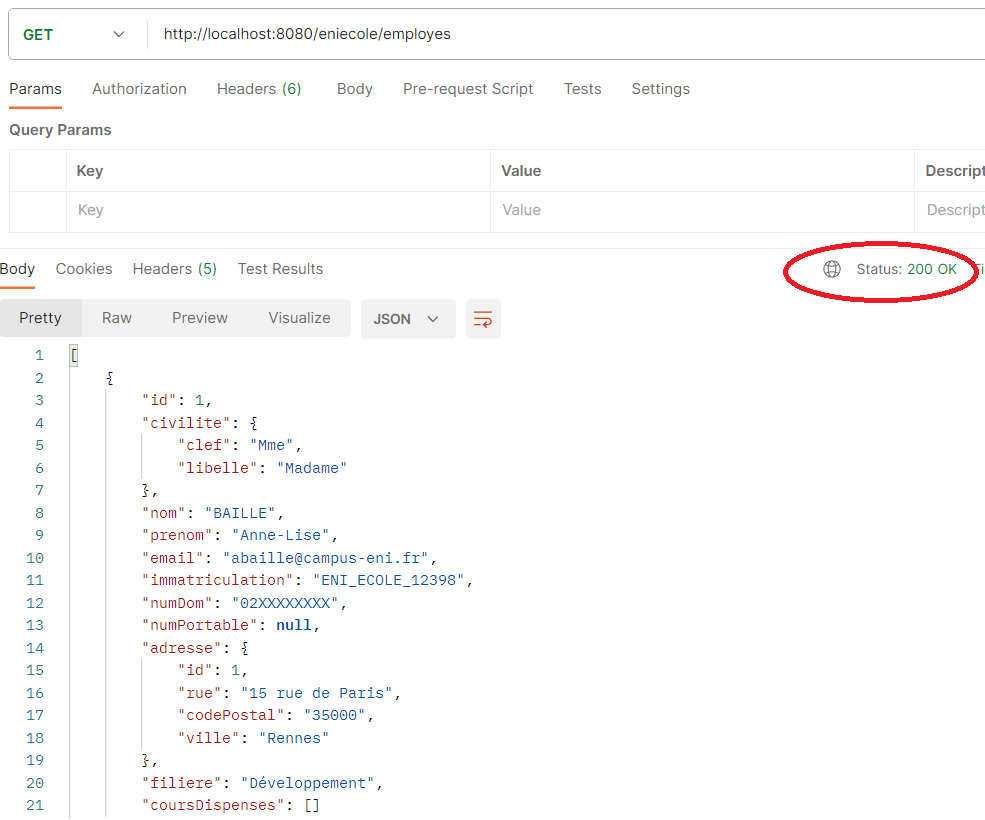
* + - Sous Chrome, l’affichage est plus minimaliste



* Nous pouvons remarquer que l’affichage diffère d’un navigateur à l’autre
* Testons dans Postman

## Exécution avec Postman

* Vous pouvez enregistrer votre requête précédente
* Cliquer pour créer une nouvelle requête
* Indiquer l’URL http://localhost:8080/eniecole/employes
* Exécuter la requête



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

# Get Mapping et gestion d’un statut NO\_CONTENT

Imaginons, que notre service retourne une liste nulle ou vide. Pour le client, il n’y a rien à afficher

* Avec le statut : 204 No Content ; nous donnons cette information au navigateur et à l’application appelante
* Pour mettre cela en place, il nous faut retourner un ResponseEntity
* Remplacer List<Employe> par ResponseEntity< ?>
  + Déclarer une variable locale pour représenter les données transmises par la couche BLL
  + Valider le résultat
    - Si aucun employé ; renvoyer un NO\_CONTENT avec le builder noContent()
    - Si des employés ; renvoyer OK et la liste des employés avec le builder ok()

@GetMapping

**public** ResponseEntity<?> rechercherTousEmployes() {

**final** List<Employe> employes = employeService.chargerTousEmployes();

**if** (employes == **null** || employes.isEmpty()) {

// Statut 204 : No Content - Pas de body car rien à afficher

**return** ResponseEntity.*noContent*().build();

}

// Statut 200 : OK + dans le body employes

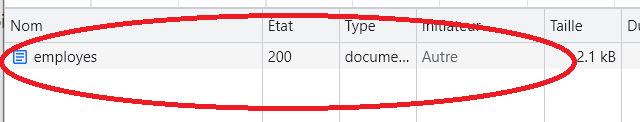
// Le contenu du body est directement injecté dans la méthode ok

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

}

## Exécution dans un navigateur

* + Ouvrez un navigateur et saisir l’URL : http://localhost:8080/eniecole/employes
  + Vous obtiendrez, le même résultat que précédemment
    - Le tableau d’Employe au format JSON
    - Pour voir que le statut est 200, il faut activer les outils de développement



## Exécution avec Postman

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

* Vous voyez les données en JSON – c’est toujours le même type d’affichage
* Le statut de la réponse (200)

Dans la suite des démonstrations, nous allons utiliser par défaut Postman.

* Mettre la liste à vide pour valider l’envoie du NO\_CONTENT

@GetMapping

**public** ResponseEntity<?> rechercherTousEmployes() {

**final** List<Employe> employes = **new** ArrayList<>();//employeService.chargerTousEmployes();

**if** (employes == **null** || employes.isEmpty()) {

// Statut 204 : No Content - Pas de body car rien à afficher

**return** ResponseEntity.*noContent*().build();

}

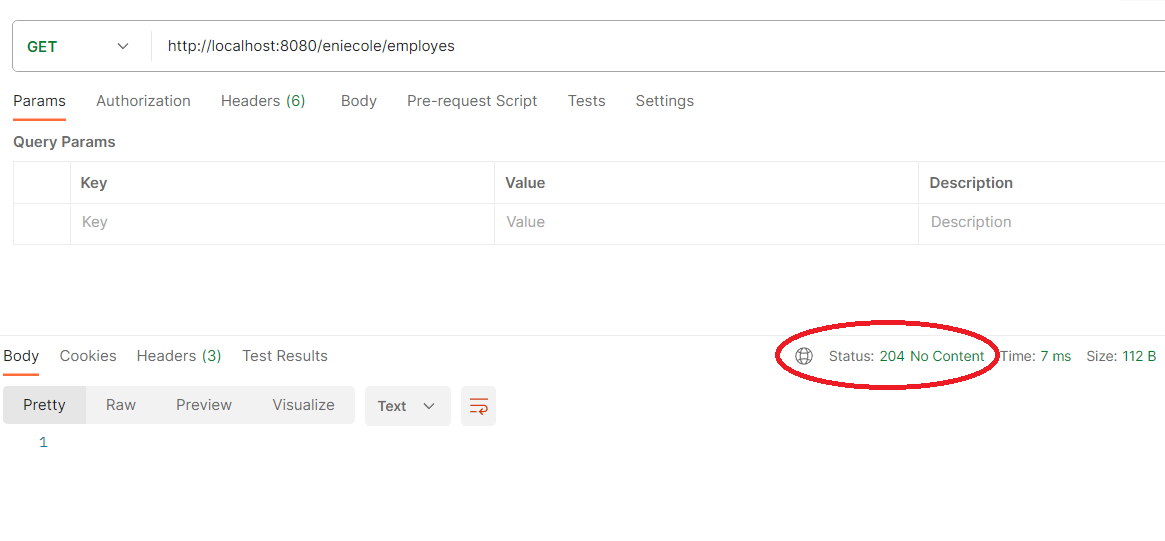
// Statut 200 : OK + dans le body employes

// Le contenu du body est directement injecté dans la méthode ok

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

}

## Exécution avec Postman



* + Il indique bien, que la réponse de la demande a un statut 204
  + Il n’affiche rien dans Body
  + Si vous essayez dans votre navigateur, il n’affiche rien. Pour être précis il ne se rafraîchit pas.
* Maintenant, que nous avons vérifié le comportemnet du ResponseEntity et de NO\_CONTENT
  + Réactiver l’appel de la méthode de la BLL dans votre code pour revenir à l’état normal

@GetMapping

**public** ResponseEntity<?> rechercherTousEmployes() {

**final** List<Employe> employes = employeService.chargerTousEmployes();

**if** (employes == **null** || employes.isEmpty()) {

// Statut 204 : No Content - Pas de body car rien à afficher

**return** ResponseEntity.*noContent*().build();

}

// Statut 200 : OK + dans le body employes

// Le contenu du body est directement injecté dans la méthode ok

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

}