

19/05/2023

Version 4



SPRING

JÉRÉMY PERROUULT

A decorative wavy line in light blue and white, running vertically along the left side of the slide.

SPRING REST

API REST

PRÉSENTATION SPRING REST

- Sommaire

- Protocol **HTTP**

- Mode client / serveur déconnecté

- Ressource Web mise à disposition

- Format de la donnée **XML** ou **JSON** (**JSON** par défaut)

- Utilisation des commandes **HTTP**

- | | | |
|----------|--|-----------------------------------|
| • GET | Obtenir une information, une ressource | Obtenir un produit |
| • POST | Transmettre des informations | Ajouter un produit |
| • PUT | Remplacer une ressource | Modifier un produit |
| • PATCH | Modifier une ressource | Modifier partiellement un produit |
| • DELETE | Supprimer une ressource | Supprimer un produit |

PRÉSENTATION SPRING REST

- Contrôleur `@Controller`
- Contrôleur REST `@RestController`

PRÉSENTATION SPRING REST

- On va communiquer (envoyer et recevoir des informations) avec le format **JSON**
 - Utilisation des dépendances
 - **jackson-databind** pour la sérialisation des objets **JAVA** en **JSON**
 - **jackson-datatype-jsr310** pour la sérialisation des objets **JAVA** Date et Time
 - **Spring** cherche automatiquement toutes seules les possibilités de sérialisation **JSON**

PRÉSENTATION SPRING REST

- Dans le contrôleur **REST**, les mappings Web s'appliquent de la même façon
 - @RequestMapping
 - @GetMapping
 - @PostMapping
 - ...
- Les signatures des méthodes ont également accès
 - @PathVariable
 - @RequestParam

```
@GetMapping("/{id}")  
public Fournisseur findById(@PathVariable int id) {  
    return this.daoFournisseur.findById(id).get();  
}
```

PRÉSENTATION SPRING REST

- Par défaut, **Spring** n'autorise pas les communications depuis un autre domaine
 - Il sera impossible d'interagir avec **AJAX** (par exemple) depuis un autre domaine
 - C'est ce qu'on appelle le **CROS** (**C**ross-**O**rigine **R**esource **S**haring)
 - Un paramètre de réponse **HTTP** "Access-Control-Allow-Origin" spécifie les autorisations
- Pour modifier ce comportement
 - Utiliser l'annotation `@CrossOrigin("*")` sur le `@RestController`

RÉCEPTION DE DONNÉES

- Outil **Postman** (Extension Chrome ou application standalone)
 - Permettra de tester les contrôleurs **REST**, en indiquant
 - La ressource (URL)
 - La méthode **HTTP** à utiliser (GET, POST, PUT, PATCH, DELETE)
 - Le corps de la requête
 - Doit être de type `application/json` si **JSON** utilisé
 - Doit être de type `application/xml` si **XML** utilisé



ENVOI

ENVOYER DES DONNÉES

ENVOI DE DONNÉES

- Pour envoyer de l'information au format **JSON**
 - Si l'annotation `@Controller` est utilisée plutôt que l'annotation `@RestController`
 - Utilisation de l'annotation `@ResponseBody` qui permet de manipuler la réponse **HTTP**
 - Sinon, rien à faire de particulier !

ENVOI DE DONNÉES

- Retourner tous les produits de la base de données
 - `http://localhost:8080/projet/api/produit` (GET)

```
@RestController
@RequestMapping("/produit")
public class ProduitRestController {
    @Autowired
    private IProduitRepository repoProduit;

    @GetMapping
    //Si utilisation de @Controller au lieu de @RestController
    @ResponseBody
    public List<Produit> findAll() {
        return this.repoProduit.findAll();
    }
}
```

EXERCICE

- Créer un `@RestController` `ProduitRestController`
 - Retourner un produit
 - `http://localhost:8080/projet/api/produit/demo` (GET)
 - Retourne un `new Produit()`
 - **Ne pas utiliser de Repository pour le moment !**

A decorative wavy line in light blue and white, flowing vertically along the left side of the slide.

RÉCEPTION

RECEVOIR DES DONNÉES

RÉCEPTION DE DONNÉES

- Pour recevoir de l'information au format **JSON**
 - Utilisation de l'annotation `@RequestBody`
 - Permet de Binder le corps de la requête à un objet, ou une liste d'objets

RÉCEPTION DE DONNÉES

- Insérer un produit et le retourner
 - <http://localhost:8080/projet/api/produit> (POST)

```
@PostMapping
public Produit add(@RequestBody Produit produit) {
    this.repoProduit.save(produit);
    return produit;
}
```

RÉCEPTION DE DONNÉES

- Insérer un produit et le retourner
 - Les validateurs peuvent être utilisés !

```
@PostMapping
public Produit add(@Valid @RequestBody Produit produit, BindingResult result) {
    if (result.hasErrors()) {
        throw new ProduitValidationException();
    }

    this.repoProduit.save(produit);
    return produit;
}
```


RÉCEPTION DE DONNÉES

- L'exception jetée est de type `RuntimeException`
 - Permet de ne pas interrompre le traitement avec un `throws Throwable`

```
@ResponseStatus(value = HttpStatus.BAD_REQUEST, reason = "Le produit n'a pas pu être validé")
public class ProduitValidationException extends RuntimeException {
    private static final long serialVersionUID = 1L;
}
```

RÉCEPTION DE DONNÉES

- Il faut savoir qu'en modification, c'est un annule et remplace des informations
 - Les données qui ne sont pas reçues sont modifiées et prennent la nouvelle valeur « null »

```
@PutMapping("/{id}")  
public Produit edit(@RequestBody Produit produit, @PathVariable int id) {  
    this.repoProduit.save(produit);  
    return this.repoProduit.findById(produit.getId()).get();  
}
```

- Il faut donc penser à donner toutes les informations, mêmes celles qui ne sont pas à modifier ...

RÉCEPTION DE DONNÉES

- ... ou prévoir un traitement qui modifie partiellement

```
@PatchMapping("/{id}")
public Produit partialEdit(@RequestBody Map<String, Object> fields, @PathVariable int id) {
    Produit produit = this.repoProduit.findById(id).get();

    fields.forEach((key, value) -> {
        Field field = ReflectionUtils.findField(Produit.class, key);
        ReflectionUtils.makeAccessible(field);
        ReflectionUtils.setField(field, produit, value);
    });

    this.repoProduit.save(produit);
    return produit;
}
```

RÉCEPTION DE DONNÉES

- Et dans le cas où on cherche à valider l'objet avant modification
 - Pas besoin de se poser la question ...
 - Il faut toutes les informations pour que la validation se fasse correctement !

EXERCICE

- Dans le `@RestController ProduitRestController`
 - Ajouter un produit
 - `http://localhost:8080/projet/api/produit` (POST)
 - Attend un produit en paramètre
 - Vérifier et tester avec des `System.out.println()`
 - **Ne pas utiliser de Repository pour le moment !**

EXERCICE

- Dans le `@RestController ProduitRestController`
 - Modifier la méthode d'ajout d'un produit
 - `http://localhost:8080/projet/api/produit` (POST)
 - Attend un produit **valide** en paramètre
 - **Utiliser la Repository !**

A decorative graphic on the left side of the slide consisting of two parallel, wavy vertical lines. The inner line is a light blue color, and the outer line is white. They start from the top left and extend towards the bottom left, creating a stylized, organic shape.

JACKSON

PROBLÉMATIQUES JACKSON

PROBLÉMATIQUES JACKSON

- Une boucle sans fin
 - Un produit a un fournisseur
 - Un fournisseur a une liste de produits
 - Chaque produit a un fournisseur
 - Le fournisseur de chaque produit a une liste de produits
 - ... boucle infinie ...

PROBLÉMATIQUES JACKSON

- Utilisation d'une annotation de **jackson-databind** dans le modèle (selon les besoins)
 - @JsonIgnore sur les attributs à ignorer
 - Dans l'exemple, on peut ignorer la *liste des produits* pour la classe Fournisseur
 - @JsonBackReference sur les attributs de retour
 - Dans l'exemple, la référence de retour est l'attribut *fournisseur* de la classe Produit
 - @JsonIgnoreProperties sur les attributs circulaires
 - Dans l'exemple
 - @JsonIgnoreProperties("fournisseur") sur l'attribut *produits* de la classe Fournisseur
 - @JsonIgnoreProperties("produits") sur l'attribut *fournisseur* de la classe Produit

PROBLÉMATIQUES JACKSON

- Utilisation d'une classe (ou interface) de « projection »
- Utilisation d'une annotation de **jackson-databind** dans le modèle et le contrôleur
 - @JsonView sur les attributs modèle et sur la méthode du contrôleur

```
public interface Views {  
    public static interface Common { }  
  
    public static interface Produit extends Common { }  
  
    public static interface ProduitWithAchats extends Produit { }  
}
```

PROBLÉMATIQUES JACKSON

- Classe modèle

```
public class Produit {  
    @JsonView( Views.Common.class)  
    private id id;  
  
    @JsonView( Views.Produit.class)  
    private String libelle;  
  
    @JsonView( Views.ProduitWithAchats.class)  
    private List<Achat> achats;  
}
```

- Classe contrôleur

```
public class ProduitController {  
    @JsonView( Views.Produit.class)  
    public List<Produit> findAll() {  
        return this.repoProduit.findAll();  
    }  
  
    @JsonView( Views.ProduitWithAchats.class)  
    public Produit findById(int id) {  
        return this.repoProduit.findByIdWithAchats(id);  
    }  
}
```

- Dans le `findAll`, les attributs `id` et `libelle` sont retournés pour les produits
- Dans le `findById`, les attributs `id`, `libelle` et `achats` sont retournés pour le produit

EXERCICE

- Retourner un produit avec son fournisseur et sa liste de produits
 - `http://localhost:8080/projet/api/produit/{id}` (GET)
 - Retourne un produit par son ID, son fournisseur et sa liste de produits
 - **Utiliser la Repository !**
 - **!! Ne pas oublier !!**
 - De créer une `@Query` de jointure
 - **OU**
 - D'initialiser la liste avec `Hibernate.initialize()`, et dans ce cas, penser à
 - Activer les annotations `@Transactional` dans la configuration
 - Annoter la méthode du contrôleur de `@Transactional`

EXERCICE

- Modifier et créer des `@RestController` pour
 - Liste des produits
 - `http://localhost:8080/projet/api/produit` (GET)
 - Ajouter un produit
 - `http://localhost:8080/projet/api/produit` (POST)
 - Récupérer un produit
 - `http://localhost:8080/projet/api/produit/{id}` (GET)
 - Modifier un produit
 - `http://localhost:8080/projet/api/produit/{id}` (PUT)
 - Supprimer un produit
 - `http://localhost:8080/projet/api/produit/{id}` (DELETE)