



# SPRING

Jérémy PERROUAULT



# SPRING MVC REST

Services Web REST

# PRÉSENTATION

## Sommairement

- 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
  - POST                    Transmettre des informations
  - PUT                      Remplacer une ressource
  - PATCH                  Modifier une ressource
  - DELETE                Supprimer une ressource

Obtenir un produit

Ajouter un produit

Modifier un produit

Modifier partiellement un produit

Supprimer un produit

# PRÉSENTATION

Contrôleur

**@Controller**

Contrôleur REST

**@RestController**

# CONFIGURATION

## Déclaration d'une nouvelle DispatcherServlet

- Mappée sur /api/\*
- Nouvelle configuration
  - Nouveau fichier de configuration *api-context.xml*
  - OU
  - Nouvelle classe de configuration **ApiConfig**
  - La nouvelle configuration doit scanner le ou les package(s) dans le(s)quel(s) se trouve(nt) les RestControllers

# CONFIGURATION

Dans le fichier *web.xml*

- Si configuration par XML

```
<!-- Déclaration de la Servlet DispatcherServlet REST (String MVC) -->
<servlet>
  <servlet-name>api</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <!-- On utilise le fichier de configuration api-context.xml -->
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/api-context.xml</param-value>
  </init-param>

  <load-on-startup>2</load-on-startup>
</servlet>
```

# CONFIGURATION

Dans le fichier *web.xml*

- Si configuration par classe

```
<!-- Déclaration de la Servlet DispatcherServlet REST (String MVC) -->
<servlet>
  <servlet-name>api</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
  </init-param>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>fr.formation.config.ApiConfig</param-value>
  </init-param>

  <load-on-startup>2</load-on-startup>
</servlet>
```

# CONFIGURATION

Dans le fichier *web.xml*

- Mapping de la DispatcherServlet sur « /api/\* »

```
<!-- Mapping de api sur toutes les ressources /api/ -->  
<servlet-mapping>  
  <servlet-name>api</servlet-name>  
  <url-pattern>/api/*</url-pattern>  
</servlet-mapping>
```



# COMMUNICATION

On va communiquer (envoyer et recevoir des informations) avec le format JSON

- Utilisation de la dépendance **jackson-databind**
- Les versions récentes de Spring cherchent toutes seules les possibilités de sérialisation JSON

# COMMUNICATION

Dans le contrôleur REST, les mappings Web s'appliquent de la même façon

- **@RequestMapping**
- **@GetMapping**
- **@PostMapping**
- ...

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

Les signatures des méthodes ont également accès

- **@PathVariable**
- **@RequestParam**

# COMMUNICATION

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 (Cross-Origin Resource Sharing)
- Un paramètre de réponse HTTP « Access-Control-Allow-Origin » spécifie les autorisations de partage

Pour modifier ce comportement

- Utiliser l'annotation **@CrossOrigin("\*")** sur le **@RestController**

# ENVOYER

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 !

# ENVOYER

Retourner tous les produits de la base de données

- <http://localhost:8080/projet/api/produit> (GET)

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

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

# EXERCICE

Créer un **@RestController ProduitRestController**

- Retourner un produit
  - `http://localhost:8080/projet/api/produit/test` (GET)
  - Retourne un new **Produit()**
  - **Ne pas utiliser de DAO pour le moment !**

# RECEVOIR

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

# RECEVOIR

Insérer un produit et le retourner

- <http://localhost:8080/projet/api/produit> (POST)

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



# RECEVOIR

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.daoProduit.save(produit);
    return produit;
}
```

# RECEVOIR

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;
}
```

# RECEVOIR

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.daoProduit.save(produit);
    return this.daoProduit.findById(produit.getId()).get();
}
```

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

# RECEVOIR

... ou prévoir un traitement qui modifie partiellement

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

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

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

# RECEVOIR

Et dans le cas où on cherche à valider l'objet avant modification

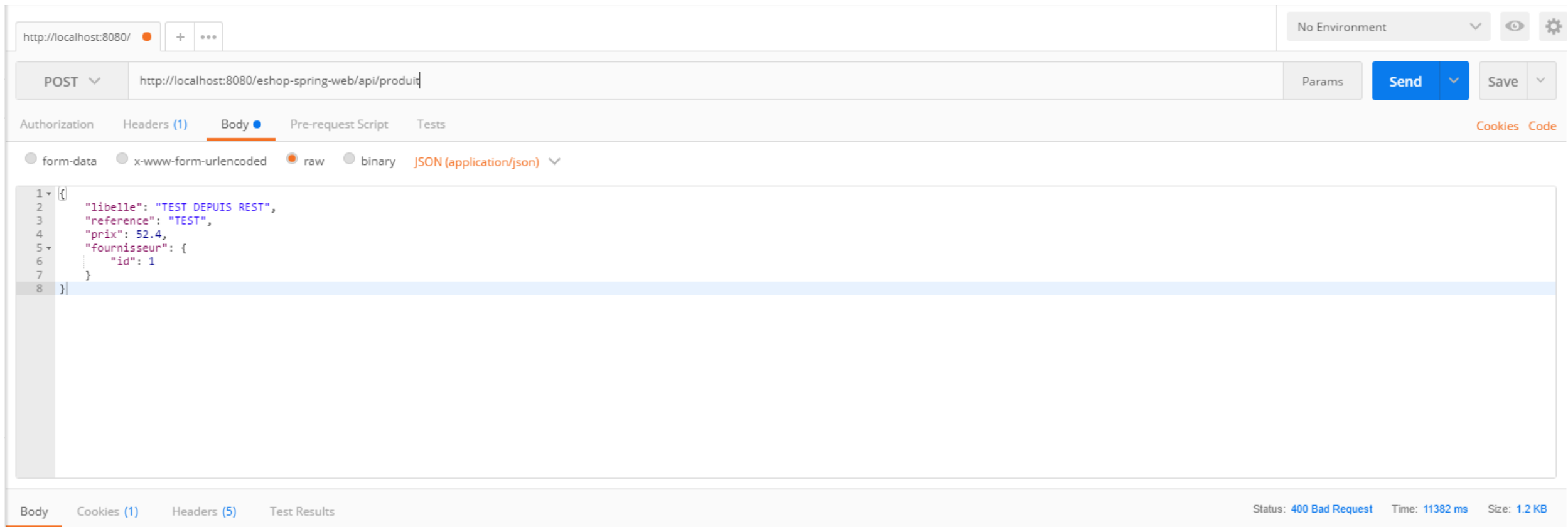
- Pas besoin de se poser la question ...
- Il faut toutes les informations pour que la validation se fasse correctement !

# OUTIL DE MANIPULATION — POSTMAN

## 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

# OUTIL DE MANIPULATION — POSTMAN



# EXERCICE

Installer Postman (et tester avec Postman)

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 DAO 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 DAO !**

# PROBLÉMATIQUE — LAZY LOADING

Spring va chercher à sérialiser tous les attributs

- Même ceux qui ne sont pas chargés (Lazy Loading) !

Modifions son comportement

- Ajout de la dépendance **jackson-datatype-hibernate5**
- Dans la configuration de l'API
  - Déclaration d'un MessageConverter
  - Ajout de ce MessageConverter dans la liste des convertisseurs de message de Spring

# PROBLÉMATIQUE — LAZY LOADING

```
<context:component-scan base-package="fr.formation.restcontroller" />

<mvc:annotation-driven>
  <mvc:message-converters>
    <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
      <property name="objectMapper">
        <bean class="fr.formation.config.HibernateObjectMapper" />
      </property>
    </bean>
  </mvc:message-converters>
</mvc:annotation-driven>
```

# PROBLÉMATIQUE — LAZY LOADING

```
public class HibernateObjectMapper extends ObjectMapper {  
    private static final long serialVersionUID = 1L;  
  
    public HibernateObjectMapper() {  
        this.registerModule(new Hibernate5Module());  
    }  
}
```

# PROBLÉMATIQUE — LAZY LOADING

```
@Configuration
@EnableWebMvc
@ComponentScan("fr.formation.restcontroller")
public class ApiConfig implements WebMvcConfigurer {
    public MappingJackson2HttpMessageConverter jsonConverter() {
        MappingJackson2HttpMessageConverter jsonConverter = new MappingJackson2HttpMessageConverter();
        ObjectMapper objectMapper = jsonConverter.getObjectMapper();

        objectMapper.registerModule(new Hibernate5Module());
        return jsonConverter;
    }

    @Override
    public void configureMessageConverters(List<HttpMessageConverter<?>> converters) {
        converters.add(this.jsonConverter());
    }
}
```

# EXERCICE

Dans le **@RestController ProduitRestController**

- Retourner la liste des produits
  - <http://localhost:8080/projet/api/produit> (GET)
  - Retourne la liste des produits
  - **Utiliser la DAO !**

# PROBLÉMATIQUE — RÉFÉRENCES CIRCULAIRES

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 ...

# PROBLÉMATIQUE — RÉFÉRENCES CIRCULAIRES

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**
  - Cette option est à privilégier !



# PROBLÉMATIQUE — RÉFÉRENCES CIRCULAIRES

Utilisation d'une classe 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 class Views {  
    public static class Common {  
    }  
  
    public static class Produit extends Common {  
    }  
  
    public static class ProduitWithAchats extends Produit {  
    }  
}
```

# PROBLÉMATIQUE — RÉFÉRENCES CIRCULAIRES

## 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.daoProduit.findAll();  
    }  
  
    @JsonView(Views.ProduitWithAchats.class)  
    public Produit findById(int id) {  
        return this.daoProduit.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 DAO !**
- **/!\ 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)