Compt Rendu

5ème année
Ingénieur Informatique & Réseaux

TP2

REST avec Spring BOOT et Spring Rest

Réalisé par : *IMANE Chakrellah*
YASSINE Ech-chaoui

Prof : *Pr.Oussama*
Classe: *5IIR-11*

2025-2026

Table des matières

Tout les TPs Réalisés ce trouve dans le lien suivant qui contient le projet + CR :

https://github.com/CHAKRELLAH44/JEE-ARCHITECTURE.git

Objectif du TP

L'objectif de ce TP est de mettre en place un service web respectant l'architecture REST à l'aide de Spring Boot.
Il s'agit de comprendre comment :

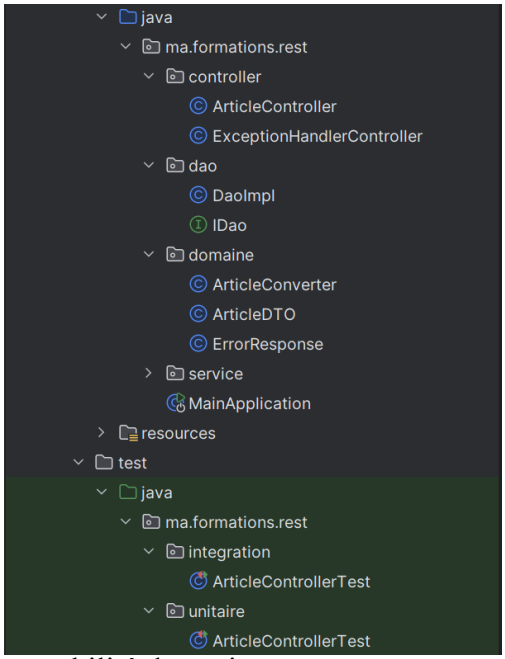
- définir et exposer des endpoints HTTP,
- structurer une application en couches (Controller, Service, DAO),
- manipuler des objets métier (Article) avec des DTO et un convertisseur,
- gérer les règles métier et les validations,
- gérer les erreurs avec un système d'exception global,
- tester l'application à l'aide de Postman et de tests JUnit.

L'enjeu principal est d'acquérir une vision claire du fonctionnement interne d'une API REST : comment une requête envoyée par un client traverse les différentes couches, comment le traitement est effectué et comment la réponse est renvoyée au format JSON ou XML.

1. Architecture du projet

1.1 Structure du projet

Le projet est structuré en plusieurs packages, chacun ayant un rôle fonctionnel bien défini.
Cette organisation permet de séparer proprement les responsabilités.



- **controller/** : expose les endpoints REST accessibles aux clients.
- **service/** : contient la logique métier, c'est-à-dire les règles et opérations appliquées aux données.
- **dao/** : simule une couche d'accès aux données (base en mémoire).
- **domaine/** : regroupe les DTO, convertisseurs et objets envoyés au client.
- **exception/** : contient les classes liées à la gestion des erreurs métier.
- **tests/** : classes de tests unitaires et intégration.

Cette structure reflète le pattern **MVC étendu**, où chaque couche a des responsabilités bien séparées afin d'assurer la maintenabilité du projet.

1.2 Description des composants

Controller

Le contrôleur est la partie visible du service REST : il reçoit les requêtes HTTP, appelle les services nécessaires et renvoie une réponse formatée.

Service

La couche service implémente toutes les règles métier nécessaires avant d'enregistrer ou de renvoyer un article.
Elle vérifie par exemple :

- si un article existe,
- si les données envoyées sont correctes,
- si des valeurs doivent être ajustées ou validées.

DAO

La couche DAO simule une petite base de données en utilisant une `ArrayList`.
Elle fournit des méthodes pour retrouver, ajouter, modifier ou supprimer des articles.

-DTO & Converter

Les DTO permettent de contrôler exactement les données envoyées au client.
Le convertisseur transforme les objets métier (Article) en DTO, et inversement.

-Exception Handler

`ExceptionHandlerController` permet de capturer toutes les erreurs de l'application pour renvoyer au client des messages clairs et structurés.

1.3 Rôle de chaque couche

- **Controller** : interface entre le client et l'application.
- **Service** : cœur du traitement des données.
- **DAO** : accès aux données (ici simulé).
- **DTO/Converter** : sert à isoler la logique REST du modèle interne.
- **Exception Handler** : centralise la gestion des erreurs.

Cette séparation clarifie le fonctionnement général : chaque couche a une responsabilité unique, ce qui améliore la lisibilité et la maintenabilité.

2. Notions importantes

2.1 Concepts théoriques

API REST

REST repose sur l'utilisation des verbes HTTP (GET, POST, PUT, DELETE) pour manipuler des ressources.
Chaque endpoint correspond à une action claire sur une ressource .

Validation

Spring Boot gère la validation automatique des données grâce aux annotations (@Size, @Min…).
Si les données sont invalides, une erreur 400 est renvoyée automatiquement.

DTO

Les Data Transfer Objects transportent des données entre les couches sans exposer les classes internes.

Exception Handling global

Avec `@ControllerAdvice`, l'application capte et renvoie des erreurs propres, compréhensibles et formatées.

2.2 Technologies utilisées

- Java 17
- Spring Boot 3
- Spring Web
- Spring Validation (Bean Validation)
- Lombok
- Maven
- JUnit 5
- Postman pour les tests
- Tomcat embarqué (servlet container)

2.3 Principes clés

- Architecture en couches

L'architecture en couches consiste à organiser une application en plusieurs niveaux indépendants, où chaque couche a une responsabilité précise.
Dans notre TP, l'application suit généralement **trois couches principales** Controller-Service-DAO

- Injection de dépendances via Spring

L'injection de dépendances est un principe clé de Spring.
L'idée est simple : **au lieu que les classes créent directement leurs dépendances**, Spring se charge de les fournir automatiquement.

Dans `ArticleController`, on ne crée pas `ServiceImpl` avec un `new`.
On écrit juste :

```
@Autowired
private IService service;
```

- Séparation entre BO (business objects) et DTO
- Standardisation des réponses d'erreurs (Exeption)
- Tests unitaires et intégration

3. Déroulement du TP

3.1 Étapes réalisées

1. **Création du projet** via Spring Initializr avec les dépendances Spring Web + Validation + Lombok.
2. **Implémentation du modèle Article** (DTO + BO).
3. **Création des classes DAO** pour simuler une base de données en mémoire.
4. **Création de la couche service**, interface (`IService`) + implémentation (`ServiceImpl`).
5. **Création du contrôleur REST** exposant les endpoints CRUD.
6. **Ajout de la validation** des champs (description, quantité, prix).
7. **Implémentation du convertisseur** DTO ↔ BO.
8. **Création de l'exception BusinessException** et de la gestion globale des erreurs.
9. **Tests avec Postman** de tous les endpoints en JSON et XML.
10. **Écriture des tests JUnit** pour vérifier les endpoints.
11. **Génération du JAR ou WAR** pour le déploiement.

4. Code source

4.1 Extraits de code essentiels

Exemple de controller :

```
@GetMapping(value = "/all", produces = {MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_VALUE})
public List<ArticleDTO> getAll() {

    return service.getAll();
}
```

Exemple de validation :

```
@Size(min = 1, max = 30, message = "description size must be between 1 and 30")
private String description;
```

Exemple d'exception handler :

```
@ExceptionHandler(Exception.class)  imanska
public final ResponseEntity<Object> handleOtherExceptions(Exception ex, WebRequest request) {
    List<String> details = new ArrayList<>();
    details.add(ex.getMessage());
    ErrorResponse error = new ErrorResponse( message: "Technical error, please consult your administrator", details);
    return new ResponseEntity(error, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

4.2 Explication du workflow

Voici comment une requête circule dans l'application :

1. Le client envoie une requête HTTP (ex : POST /api/articles/create).
2. Le **Controller** reçoit la requête, vérifie la validité des données et invoque la couche service.
3. La **ServiceImpl** applique les règles métier (exemple : vérifier si l'article existe déjà).
4. La **DAO** est sollicitée pour manipuler la base mémoire.
5. Le résultat remonte vers le service puis vers le controller.
6. La réponse est renvoyée au format JSON ou XML selon la demande.
7. Si une erreur survient, elle est interceptée par le **ExceptionHandler** qui renvoie un message clair au client.

4.3 Rôle des principales classes

- **ArticleDTO / Article** : structure des données.
- **ArticleConverter** : transformation interne ↔ externe.
- **IService / ServiceImpl** : logique métier.
- **IDao / DaoImpl** : stockage.
- **ArticleController** : exposition des endpoints.
- **ExceptionHandlerController** : centralisation des erreurs.

5. Tests avec capture + explication

5.1 Tests des endpoints

1-GET – Liste complète

URL : `http://localhost:7777/api/articles/all`
Explication : récupère la liste entière des articles.


```
▼<List>
  <script id="eppiocemhmnlbhjplcgkofciiegomcon"/>
  <script/>
  <script/>
 ▼<item>
    <id>1</id>
    <description>PC PORTABLE HP I7</description>
    <price>15000.0</price>
    <quantity>10.0</quantity>
  </item>
 ▼<item>
    <id>2</id>
    <description>ECRAN</description>
    <price>1500.0</price>
    <quantity>10.0</quantity>
  </item>
 ▼<item>
    <id>3</id>
    <description>CAMERA LG</description>
    <price>3000.0</price>
    <quantity>10.0</quantity>
  </item>
 ▼<item>
    <id>4</id>
    <description>SOURIS</description>
    <price>200.0</price>
    <quantity>10.0</quantity>
  </item>
</List>
```

2-GET – Article par ID (PathVariable)

URL : `http://localhost:7777/api/articles/id/1`
Explication : renvoie l'article dont l'ID est 1.

IPC PORTABLE HP I715000.010.0

3-GET – Article par ID (RequestParam)

URL : `http://localhost:7777/api/articles?id=1`
Explication : même fonctionnalité mais avec paramètre URL.


```
{
    "id": 1,
    "description": "PC PORTABLE HP I7",
    "price": 15000.0,
    "quantity": 10.0
}
```


4- POST – Création article

URL : `http://localhost:7777/api/articles/create`
Explication : ajoute un nouvel article dans la base mémoire.


```
{
  "id": 5,
  "description": "PC FIXE",
  "price": 12000,
  "quantity": 10
}
```

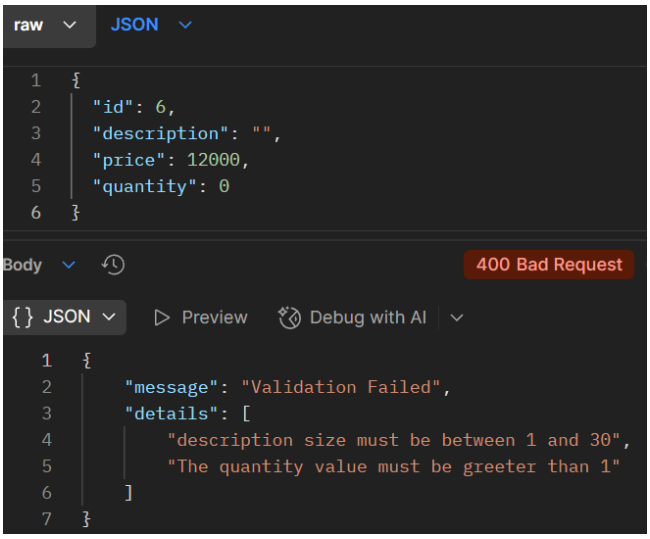

ody 201 Created

Raw Preview Visualize

```
1  Article is created successfully
```

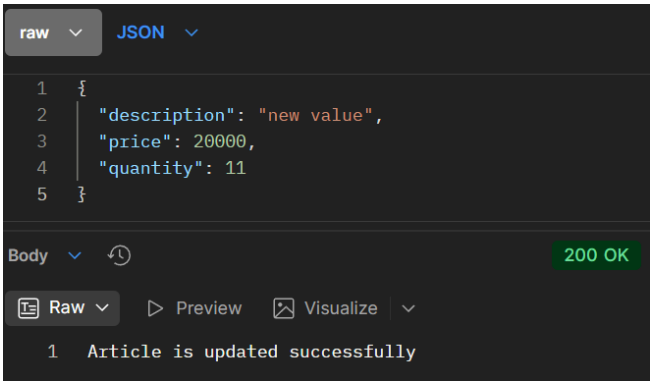
5- POST – Création article invalide

Explication : permet de tester la validation automatique et de vérifier que l'exception handler renvoie une réponse structurée.



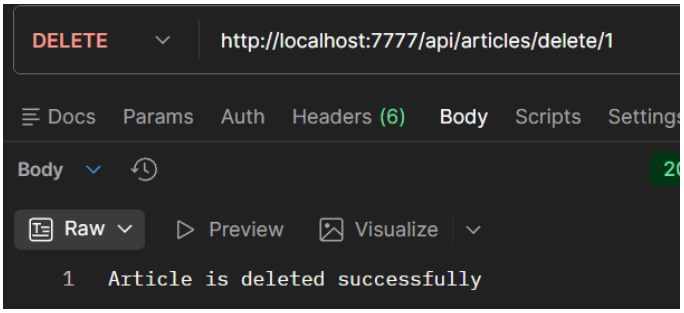
6-PUT – Mise à jour d'un article

URL : `http://localhost:7777/api/articles/update/1`
Explication : modifie l'article ayant l'ID 1.



7- DELETE – Suppression

URL : `http://localhost:7777/api/articles/delete/1`
Explication : supprime l'article de la liste en mémoire.



7. Outils de visualisation

Les outils utilisés pour valider et observer le fonctionnement :

- **Postman** : interrogation manuelle de l'API.
- **Browser / XML visualizer** : affichage des réponses XML.
- **JUnit** : exécution automatique des tests.
- **Console Spring Boot** : affichage des logs d'exécution.

8. Conclusion

Ce TP a permis de comprendre le fonctionnement d'une API REST à travers une application Spring Boot complète.
Nous avons appris à structurer un projet en couches, à exposer des endpoints, à valider les données, à gérer les erreurs proprement et à tester l'ensemble avec Postman et JUnit.

Les compétences acquises incluent :

- architecture logicielle propre et modulaire,
- maîtrise de Spring Web, DTO, validation, exception handling,
- compréhension du workflow HTTP interne dans Spring,
- bases du test d'API.

Limites du TP

- stockage en mémoire (pas de vraie base de données),
- pas d'authentification ni autorisation,
- absence de pagination ou filtrage avancé,
- couverture de test partielle.