Développement Web Full Stack

TD1 : Spring Boot

Création d'une API REST

Ibrahim ALAME

Durée : 2 heures

Spring Boot
API REST + JPA + H2

Objectifs du TD

À la fin de ce TD, vous serez capable de :

- Créer un projet Spring Boot avec les bonnes dépendances
- Définir des entités JPA avec les annotations appropriées
- Créer des repositories pour l'accès aux données
- Implémenter un contrôleur REST avec les opérations CRUD
- Tester votre API avec curl ou Postman
- Configurer et utiliser la base de données H2

Prérequis :

- JDK 17 ou supérieur installé
- IntelliJ IDEA (ou un autre IDE Java)
- Postman ou curl pour tester l'API

Contexte

Vous allez développer une API REST pour gérer une **bibliothèque de livres**. L'API permettra de :

- Lister tous les livres
- Récupérer un livre par son ID
- Ajouter un nouveau livre
- Modifier un livre existant
- Supprimer un livre

1 Création du projet (15 min)

Exercice 1 : Initialisation du projet Spring Boot

> **Objectif :** Créer un nouveau projet Spring Boot avec les dépendances nécessaires.

Étapes à suivre

1. Aller sur **Spring Initializr** : https://start.spring.io
2. Configurer le projet :
   - **Project :** Maven
   - **Language :** Java
   - **Spring Boot :** 3.2.x (dernière version stable)
   - **Group :** `com.bibliotheque`
   - **Artifact :** `livres-api`
   - **Name :** `livres-api`
   - **Packaging :** Jar
   - **Java :** 17
3. Ajouter les dépendances :
   - **Spring Web** : Pour créer l'API REST
   - **Spring Data JPA** : Pour la persistance des données
   - **H2 Database** : Base de données en mémoire pour le développement
4. Cliquer sur **Generate** pour télécharger le projet
5. Extraire l'archive et ouvrir le projet dans IntelliJ IDEA

Vérification

Ouvrez le fichier `pom.xml` et vérifiez que les dépendances suivantes sont présentes :

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
```

```
</dependencies>
```

Listing 1: Dépendances dans pom.xml

2 Configuration de la base de données (10 min)

Exercice 2 : Configuration de H2

> **Objectif :** Configurer la base de données H2 pour le développement.

Créez ou modifiez le fichier `src/main/resources/application.properties` :

```
# Configuration de la base de donnees H2
spring.datasource.url=jdbc:h2:file:./data/bibliotheque;AUTO_SERVER=TRUE
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Configuration JPA
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# Nom de l'application
spring.application.name=livres-api
```

Listing 2: application.properties

Explication des propriétés :

- `AUTO_SERVER=TRUE` : Permet plusieurs connexions simultanées (IntelliJ + application)
- `ddl-auto=update` : Crée/met à jour automatiquement les tables
- `show-sql=true` : Affiche les requêtes SQL dans la console

Test

Lancez l'application (clic droit sur la classe principale → Run). Vous devriez voir dans la console :

```
Started LivresApiApplication in X.XXX seconds
```

3 Création de l'entité Livre (20 min)

Exercice 3 : Définition de l'entité JPA

> **Objectif :** Créer l'entité `Livre` qui représente la table dans la base de données.

Spécifications

L'entité `Livre` doit avoir les attributs suivants :

- `id` : Identifiant unique (Long, auto-généré)
- `titre` : Titre du livre (String, obligatoire)
- `auteur` : Nom de l'auteur (String, obligatoire)
- `isbn` : Numéro ISBN (String, unique)
- `anneePublication` : Année de publication (Integer)
- `disponible` : Indique si le livre est disponible (boolean, défaut: true)

À faire

Créez la classe `Livre` dans le package `com.bibliotheque.livresapi.model` :

```java
package com.bibliotheque.livresapi.model;

import jakarta.persistence.*;

// TODO: Ajouter les annotations JPA appropriees

public class Livre {

    // TODO: Definir l'identifiant avec generation automatique
    private Long id;

    // TODO: Ajouter les contraintes (non null, unique pour
       isbn)
    private String titre;
    private String auteur;
    private String isbn;
    private Integer anneePublication;
    private boolean disponible = true;

    // TODO: Creer les constructeurs (vide + avec parametres)

    // TODO: Generer les getters et setters
}
```

Listing 3: model/Livre.java - À compléter

Annotations à utiliser

Annotation	Utilité
`@Entity`	Indique que la classe est une entité JPA
`@Table(name = "...")`	Définit le nom de la table (optionnel)
`@Id`	Marque le champ comme clé primaire
`@GeneratedValue`	Génération automatique de l'ID
`@Column(nullable = false)`	Colonne obligatoire
`@Column(unique = true)`	Valeur unique

> **Solution :** Voir l'annexe A à la fin du TD.

4 Création du Repository (10 min)

Exercice 4 : Définition du Repository

> **Objectif :** Créer l'interface repository pour accéder aux données.

Spring Data JPA génère automatiquement l'implémentation des méthodes CRUD de base. Il suffit de créer une interface qui étend `JpaRepository`.

À faire

Créez l'interface `LivreRepository` dans le package `com.bibliotheque.livresapi.repository` :

```java
package com.bibliotheque.livresapi.repository;

import com.bibliotheque.livresapi.model.Livre;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.util.Optional;

@Repository
public interface LivreRepository extends JpaRepository<Livre, Long> {

    // Methodes generees automatiquement par Spring Data JPA :
    // - findAll() : List<Livre>
    // - findById(Long id) : Optional<Livre>
    // - save(Livre livre) : Livre
    // - deleteById(Long id) : void
    // - existsById(Long id) : boolean

    // TODO: Ajouter des methodes de recherche personnalisees
    // Exemple : trouver par auteur, par titre contenant un mot, etc.

}
```

```
}
```

Listing 4: repository/LivreRepository.java

Méthodes personnalisées à ajouter

Ajoutez les signatures de méthodes suivantes (Spring Data JPA génère l'implémentation automatiquement) :

1. Trouver tous les livres d'un auteur donné
2. Trouver un livre par son ISBN
3. Trouver tous les livres disponibles
4. Trouver les livres dont le titre contient un mot-clé

> **Solution :** Voir l'annexe A à la fin du TD.

5 Création du Contrôleur REST (35 min)

Exercice 5 : Implémentation des endpoints CRUD

> **Objectif :** Créer le contrôleur REST avec toutes les opérations CRUD.

Endpoints à implémenter

Méthode	URL	Description
GET	`/api/livres`	Récupérer tous les livres
GET	`/api/livres/{id}`	Récupérer un livre par ID
POST	`/api/livres`	Créer un nouveau livre
PUT	`/api/livres/{id}`	Modifier un livre existant
DELETE	`/api/livres/{id}`	Supprimer un livre

À faire

Créez la classe `LivreController` dans le package `com.bibliotheque.livresapi.controller` :

```
package com.bibliotheque.livresapi.controller;

import com.bibliotheque.livresapi.model.Livre;
import com.bibliotheque.livresapi.repository.LivreRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```java

@RestController
@RequestMapping("/api/livres")
public class LivreController {

    @Autowired
    private LivreRepository livreRepository;

    // TODO: Implementer GET /api/livres

    // TODO: Implementer GET /api/livres/{id}

    // TODO: Implementer POST /api/livres

    // TODO: Implementer PUT /api/livres/{id}

    // TODO: Implementer DELETE /api/livres/{id}
}
```

Listing 5: controller/LivreController.java - Structure de base

Indications pour chaque méthode

GET /api/livres

```java
@GetMapping
public List<Livre> getAllLivres() {
    // Utiliser livreRepository.findAll()
}
```

GET /api/livres/{id}

```java
@GetMapping("/{id}")
public ResponseEntity<Livre> getLivreById(@PathVariable Long
    id) {
    // Utiliser livreRepository.findById(id)
    // Retourner 404 si non trouve
}
```

POST /api/livres

```java
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public Livre createLivre(@RequestBody Livre livre) {
    // Utiliser livreRepository.save(livre)
}
```

PUT /api/livres/{id}

```java
@PutMapping("/{id}")
public ResponseEntity<Livre> updateLivre(
        @PathVariable Long id,
        @RequestBody Livre livreDetails) {
    // Trouver le livre existant
```

```
    // Mettre a jour ses proprietes
    // Sauvegarder et retourner
    // Retourner 404 si non trouve
}
```

DELETE /api/livres/{id}

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteLivre(@PathVariable Long id) {
    // Verifier si le livre existe
    // Supprimer et retourner 204 No Content
    // Retourner 404 si non trouve
}
```

Solution : Voir l'annexe A à la fin du TD.

6 Tests de l'API (30 min)

Exercice 6 : Tester les endpoints avec curl

Objectif : Vérifier que tous les endpoints fonctionnent correctement.

Lancez votre application Spring Boot, puis testez chaque endpoint.

1. Créer des livres (POST)

```
curl -X POST http://localhost:8080/api/livres \
  -H "Content-Type: application/json" \
  -d '{
    "titre": "Le Petit Prince",
    "auteur": "Antoine de Saint-Exupery",
    "isbn": "978-2-07-040850-4",
    "anneePublication": 1943,
    "disponible": true
  }'
```

Listing 6: Créer un premier livre

```
curl -X POST http://localhost:8080/api/livres \
  -H "Content-Type: application/json" \
  -d '{
    "titre": "Les Miserables",
    "auteur": "Victor Hugo",
    "isbn": "978-2-07-040851-1",
    "anneePublication": 1862,
    "disponible": true
  }'
```

Listing 7: Créer un deuxième livre

```
curl -X POST http://localhost:8080/api/livres \
  -H "Content-Type: application/json" \
  -d '{
    "titre": "Notre-Dame de Paris",
    "auteur": "Victor Hugo",
    "isbn": "978-2-07-040852-8",
    "anneePublication": 1831,
    "disponible": false
  }'
```

Listing 8: Créer un troisième livre

2. Lister tous les livres (GET)

```
curl http://localhost:8080/api/livres
```

Listing 9: Récupérer tous les livres

Résultat attendu : Un tableau JSON avec les 3 livres créés.

3. Récupérer un livre par ID (GET)

```
curl http://localhost:8080/api/livres/1
```

Listing 10: Récupérer le livre avec l'ID 1

```
curl -v http://localhost:8080/api/livres/999
```

Listing 11: Tester avec un ID inexistant

Résultat attendu : Statut 404 Not Found pour l'ID 999.

4. Modifier un livre (PUT)

```
curl -X PUT http://localhost:8080/api/livres/1 \
  -H "Content-Type: application/json" \
  -d '{
    "titre": "Le Petit Prince",
    "auteur": "Antoine de Saint-Exupery",
    "isbn": "978-2-07-040850-4",
    "anneePublication": 1943,
    "disponible": false
  }'
```

Listing 12: Modifier la disponibilité du livre 1

5. Supprimer un livre (DELETE)

```
curl -X DELETE http://localhost:8080/api/livres/2
```

Listing 13: Supprimer le livre avec l'ID 2

```
curl http://localhost:8080/api/livres
```

Listing 14: Vérifier la suppression

Résultat attendu : Seulement 2 livres dans la liste.

Questions de vérification

1. Quel code de statut HTTP recevez-vous lors de la création d'un livre ?
2. Que se passe-t-il si vous essayez de créer deux livres avec le même ISBN ?
3. Quel code de statut recevez-vous après une suppression réussie ?

7 Exercice bonus (si le temps le permet)

Exercice 7 : Endpoints de recherche

> **Objectif :** Ajouter des endpoints de recherche personnalisés.

Ajoutez les endpoints suivants à votre contrôleur :

1. `GET /api/livres/auteur/{auteur}` : Livres par auteur
2. `GET /api/livres/disponibles` : Livres disponibles uniquement
3. `GET /api/livres/recherche?titre={mot}` : Recherche par titre

```
// Livres par auteur
@GetMapping("/auteur/{auteur}")
public List<Livre> getLivresByAuteur(@PathVariable String
    auteur) {
     return livreRepository.findByAuteur(auteur);
}

// Livres disponibles
@GetMapping("/disponibles")
public List<Livre> getLivresDisponibles() {
     return livreRepository.findByDisponibleTrue();
}

// Recherche par titre
@GetMapping("/recherche")
public List<Livre> rechercherParTitre(@RequestParam String
    titre) {
     return
        livreRepository.findByTitreContainingIgnoreCase(titre);
}
```

Listing 15: Endpoints bonus à ajouter

Tests des endpoints bonus

```
# Livres de Victor Hugo
curl "http://localhost:8080/api/livres/auteur/Victor%20Hugo"

# Livres disponibles
curl http://localhost:8080/api/livres/disponibles

# Recherche par titre
curl "http://localhost:8080/api/livres/recherche?titre=prince"
```

Extension : Pour aller plus loin

Extension avancée

Pour les étudiants ayant terminé en avance

Cette extension vous permettra d'approfondir vos connaissances en ajoutant :

- Une **couche Service** pour séparer la logique métier
- Une **deuxième entité** (Catégorie) avec une relation
- Une **gestion des exceptions** personnalisée

Extension 1 : Couche Service (20 min)

> **Objectif :** Ajouter une couche service entre le contrôleur et le repository pour respecter l'architecture en couches.

Pourquoi une couche Service ?

- **Séparation des responsabilités** : Le contrôleur gère HTTP, le service gère la logique métier
- **Réutilisabilité** : La logique métier peut être appelée depuis plusieurs contrôleurs
- **Testabilité** : Plus facile de tester la logique métier isolément
- **Transactions** : Les méthodes du service peuvent être transactionnelles

À faire

Créez la classe `LivreService` dans le package `com.bibliotheque.livresapi.service` :

```
package com.bibliotheque.livresapi.service;

import com.bibliotheque.livresapi.model.Livre;
import com.bibliotheque.livresapi.repository.LivreRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
@Transactional
public class LivreService {

    @Autowired
    private LivreRepository livreRepository;
```

```
18
19      public List<Livre> getAllLivres() {
20          return livreRepository.findAll();
21      }
22
23      public Optional<Livre> getLivreById(Long id) {
24          return livreRepository.findById(id);
25      }
26
27      public Livre createLivre(Livre livre) {
28          // Logique metier : verifier que l'ISBN n'existe pas
                deja
29          if (livre.getIsbn() != null &&
30              livreRepository.findByIsbn(livre.getIsbn()).isPresent())
                    {
31              throw new IllegalArgumentException(
32                  "Un livre avec cet ISBN existe deja");
33          }
34          return livreRepository.save(livre);
35      }
36
37      public Optional<Livre> updateLivre(Long id, Livre
            livreDetails) {
38          return livreRepository.findById(id)
39                  .map(livre -> {
40                      livre.setTitre(livreDetails.getTitre());
41                      livre.setAuteur(livreDetails.getAuteur());
42                      livre.setIsbn(livreDetails.getIsbn());
43                      livre.setAnneePublication(livreDetails.getAnneePublicat
44                      livre.setDisponible(livreDetails.isDisponible());
45                      return livreRepository.save(livre);
46                  });
47      }
48
49      public boolean deleteLivre(Long id) {
50          if (livreRepository.existsById(id)) {
51              livreRepository.deleteById(id);
52              return true;
53          }
54          return false;
55      }
56
57      // Methodes metier supplementaires
58      public Livre emprunterLivre(Long id) {
59          Livre livre = livreRepository.findById(id)
60                  .orElseThrow(() -> new IllegalArgumentException(
61                      "Livre non trouve"));
62
63          if (!livre.isDisponible()) {
64              throw new IllegalStateException(
65                  "Ce livre n'est pas disponible");
```

```
66          }
67
68          livre.setDisponible(false);
69          return livreRepository.save(livre);
70      }
71
72      public Livre rendreLivre(Long id) {
73          Livre livre = livreRepository.findById(id)
74                  .orElseThrow(() -> new IllegalArgumentException(
75                      "Livre non trouve"));
76
77          livre.setDisponible(true);
78          return livreRepository.save(livre);
79      }
80  }
```

Listing 16: service/LivreService.java

Modifier le contrôleur

Modifiez `LivreController` pour utiliser le service :

```
1   @RestController
2   @RequestMapping("/api/livres")
3   public class LivreController {
4
5       @Autowired
6       private LivreService livreService;  // Service au lieu de
            Repository
7
8       @GetMapping
9       public List<Livre> getAllLivres() {
10          return livreService.getAllLivres();
11      }
12
13      @GetMapping("/{id}")
14      public ResponseEntity<Livre> getLivreById(@PathVariable
            Long id) {
15          return livreService.getLivreById(id)
16                  .map(ResponseEntity::ok)
17                  .orElse(ResponseEntity.notFound().build());
18      }
19
20      @PostMapping
21      @ResponseStatus(HttpStatus.CREATED)
22      public Livre createLivre(@RequestBody Livre livre) {
23          return livreService.createLivre(livre);
24      }
25
26      // Nouveaux endpoints metier
27      @PatchMapping("/{id}/emprunter")
```

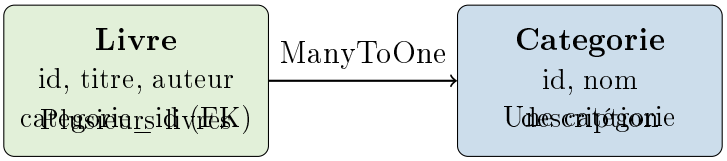
```java
    public ResponseEntity<Livre> emprunterLivre(@PathVariable Long id) {
        try {
            return ResponseEntity.ok(livreService.emprunterLivre(id));
        } catch (IllegalStateException e) {
            return ResponseEntity.badRequest().build();
        }
    }

    @PatchMapping("/{id}/rendre")
    public ResponseEntity<Livre> rendreLivre(@PathVariable Long id) {
        return ResponseEntity.ok(livreService.rendreLivre(id));
    }
}
```

Listing 17: Contrôleur modifié utilisant le service

Extension 2 : Entité Catégorie avec relation (25 min)

> **Objectif :** Créer une entité `Categorie` liée aux livres (relation ManyToOne).

Schéma de la relation



Créer l'entité Catégorie

```java
package com.bibliotheque.livresapi.model;

import jakarta.persistence.*;
import com.fasterxml.jackson.annotation.JsonIgnore;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "categories")
public class Categorie {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String nom;
```

```
18
19      private String description;
20
21      // Relation inverse (optionnel, pour naviguer de Categorie
            vers Livres)
22      @OneToMany(mappedBy = "categorie")
23      @JsonIgnore  // Evite la recursion infinie lors de la
            serialisation JSON
24      private List<Livre> livres = new ArrayList<>();
25
26      // Constructeurs
27      public Categorie() {
28      }
29
30      public Categorie(String nom, String description) {
31          this.nom = nom;
32          this.description = description;
33      }
34
35      // Getters et Setters
36      public Long getId() {
37          return id;
38      }
39
40      public void setId(Long id) {
41          this.id = id;
42      }
43
44      public String getNom() {
45          return nom;
46      }
47
48      public void setNom(String nom) {
49          this.nom = nom;
50      }
51
52      public String getDescription() {
53          return description;
54      }
55
56      public void setDescription(String description) {
57          this.description = description;
58      }
59
60      public List<Livre> getLivres() {
61          return livres;
62      }
63
64      public void setLivres(List<Livre> livres) {
65          this.livres = livres;
66      }
```

```
}
```

Listing 18: model/Categorie.java

Modifier l'entité Livre

Ajoutez la relation ManyToOne dans `Livre.java` :

```java
@Entity
@Table(name = "livres")
public class Livre {

    // ... autres champs existants ...

    @ManyToOne
    @JoinColumn(name = "categorie_id")
    private Categorie categorie;

    // Getter et Setter pour categorie
    public Categorie getCategorie() {
        return categorie;
    }

    public void setCategorie(Categorie categorie) {
        this.categorie = categorie;
    }
}
```

Listing 19: Ajout de la relation dans Livre.java

Repository et Contrôleur pour Catégorie

```java
package com.bibliotheque.livresapi.repository;

import com.bibliotheque.livresapi.model.Categorie;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface CategorieRepository extends
    JpaRepository<Categorie, Long> {
     Optional<Categorie> findByNom(String nom);
}
```

Listing 20: repository/CategorieRepository.java

```java
package com.bibliotheque.livresapi.controller;

import com.bibliotheque.livresapi.model.Categorie;
import com.bibliotheque.livresapi.model.Livre;
```

```java
import
    com.bibliotheque.livresapi.repository.CategorieRepository;
import com.bibliotheque.livresapi.repository.LivreRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/categories")
public class CategorieController {

    @Autowired
    private CategorieRepository categorieRepository;

    @Autowired
    private LivreRepository livreRepository;

    @GetMapping
    public List<Categorie> getAllCategories() {
        return categorieRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Categorie>
       getCategorieById(@PathVariable Long id) {
        return categorieRepository.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Categorie createCategorie(@RequestBody Categorie
       categorie) {
        return categorieRepository.save(categorie);
    }

    // Recuperer tous les livres d'une categorie
    @GetMapping("/{id}/livres")
    public ResponseEntity<List<Livre>> getLivresByCategorie(
            @PathVariable Long id) {
        return categorieRepository.findById(id)
                .map(cat -> ResponseEntity.ok(cat.getLivres()))
                .orElse(ResponseEntity.notFound().build());
    }
}
```

Listing 21: controller/CategorieController.java

Tests avec curl

```
# Creer une categorie "Roman"
curl -X POST http://localhost:8080/api/categories \
  -H "Content-Type: application/json" \
  -d '{"nom": "Roman", "description": "Romans et fiction"}'

# Creer une categorie "Science-Fiction"
curl -X POST http://localhost:8080/api/categories \
  -H "Content-Type: application/json" \
  -d '{"nom": "Science-Fiction", "description": "SF et
     anticipation"}'
```

Listing 22: Créer des catégories

```
curl -X POST http://localhost:8080/api/livres \
  -H "Content-Type: application/json" \
  -d '{
    "titre": "Dune",
    "auteur": "Frank Herbert",
    "isbn": "978-2-221-25807-3",
    "anneePublication": 1965,
    "disponible": true,
    "categorie": {"id": 2}
  }'
```

Listing 23: Créer un livre avec catégorie

```
curl http://localhost:8080/api/categories/2/livres
```

Listing 24: Récupérer les livres d'une catégorie

Extension 3 : Gestion des exceptions (15 min)

> **Objectif :** Créer un gestionnaire d'exceptions global pour des réponses d'erreur cohérentes.

Exception personnalisée

```
package com.bibliotheque.livresapi.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException
    {

    public ResourceNotFoundException(String message) {
        super(message);
```

```
    }

    public ResourceNotFoundException(String resource, Long id) {
        super(String.format("%s avec l'id %d non trouve",
            resource, id));
    }
}
```

Listing 25: exception/ResourceNotFoundException.java

Gestionnaire global d'exceptions

```
package com.bibliotheque.livresapi.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import
   org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<Map<String, Object>> handleNotFound(
            ResourceNotFoundException ex) {

        Map<String, Object> body = new HashMap<>();
        body.put("timestamp", LocalDateTime.now());
        body.put("status", 404);
        body.put("error", "Not Found");
        body.put("message", ex.getMessage());

        return
            ResponseEntity.status(HttpStatus.NOT_FOUND).body(body);
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<Map<String, Object>> handleBadRequest(
            IllegalArgumentException ex) {

        Map<String, Object> body = new HashMap<>();
        body.put("timestamp", LocalDateTime.now());
        body.put("status", 400);
        body.put("error", "Bad Request");
        body.put("message", ex.getMessage());

```

```
38          return
               ResponseEntity.status(HttpStatus.BAD_REQUEST).body(body);
39      }
40
41      @ExceptionHandler(IllegalStateException.class)
42      public ResponseEntity<Map<String, Object>> handleConflict(
43              IllegalStateException ex) {
44
45          Map<String, Object> body = new HashMap<>();
46          body.put("timestamp", LocalDateTime.now());
47          body.put("status", 409);
48          body.put("error", "Conflict");
49          body.put("message", ex.getMessage());
50
51          return
               ResponseEntity.status(HttpStatus.CONFLICT).body(body);
52      }
53  }
```

Listing 26: exception/GlobalExceptionHandler.java

Utilisation dans le service

```
1  public Livre getLivreByIdOrThrow(Long id) {
2      return livreRepository.findById(id)
3              .orElseThrow(() -> new
                  ResourceNotFoundException("Livre", id));
4  }
```

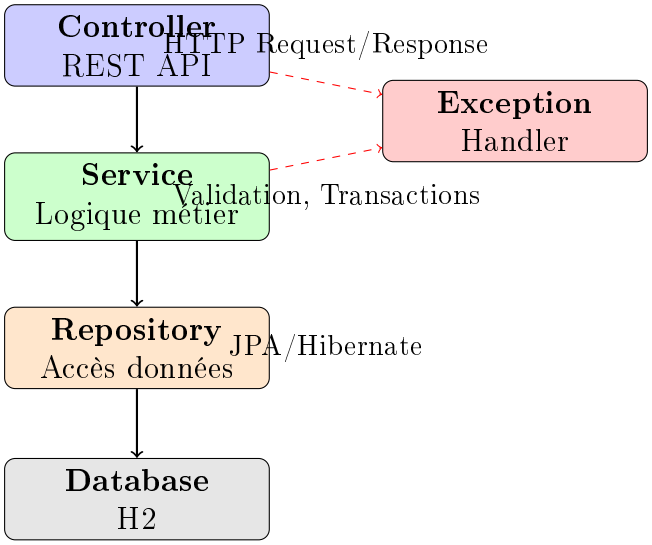
Listing 27: Utilisation de l'exception dans LivreService

Test de la gestion des erreurs

```
1  # Livre non trouve -> 404 avec message JSON
2  curl -v http://localhost:8080/api/livres/999
3
4  # ISBN duplique -> 400 avec message JSON
5  curl -X POST http://localhost:8080/api/livres \
6    -H "Content-Type: application/json" \
7    -d '{"titre": "Test", "auteur": "Test", "isbn":
       "978-2-07-040850-4"}'
```

Listing 28: Tester les erreurs

Résumé de l'architecture finale



Ce que vous avez appris dans l'extension :

- Séparer la logique métier dans une couche **Service**
- Créer des **relations entre entités** (ManyToOne / OneToMany)
- Gérer les **exceptions** de manière globale avec `@RestControllerAdvice`
- Utiliser `@Transactional` pour les opérations de base de données

Annexe A : Solutions

Solution Exercice 3 : Entité Livre

```java
package com.bibliotheque.livresapi.model;

import jakarta.persistence.*;

@Entity
@Table(name = "livres")
public class Livre {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String titre;

    @Column(nullable = false)
    private String auteur;

    @Column(unique = true)
    private String isbn;

    private Integer anneePublication;

    @Column(nullable = false)
    private boolean disponible = true;

    // Constructeur vide (requis par JPA)
    public Livre() {
    }

    // Constructeur avec parametres
    public Livre(String titre, String auteur, String isbn,
                 Integer anneePublication) {
        this.titre = titre;
        this.auteur = auteur;
        this.isbn = isbn;
        this.anneePublication = anneePublication;
        this.disponible = true;
    }

    // Getters et Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
```

```
48      }
49
50      public String getTitre() {
51          return titre;
52      }
53
54      public void setTitre(String titre) {
55          this.titre = titre;
56      }
57
58      public String getAuteur() {
59          return auteur;
60      }
61
62      public void setAuteur(String auteur) {
63          this.auteur = auteur;
64      }
65
66      public String getIsbn() {
67          return isbn;
68      }
69
70      public void setIsbn(String isbn) {
71          this.isbn = isbn;
72      }
73
74      public Integer getAnneePublication() {
75          return anneePublication;
76      }
77
78      public void setAnneePublication(Integer anneePublication) {
79          this.anneePublication = anneePublication;
80      }
81
82      public boolean isDisponible() {
83          return disponible;
84      }
85
86      public void setDisponible(boolean disponible) {
87          this.disponible = disponible;
88      }
89  }
```

Listing 29: model/Livre.java - Solution complète

Solution Exercice 4 : Repository

```java
package com.bibliotheque.livresapi.repository;

import com.bibliotheque.livresapi.model.Livre;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.util.Optional;

@Repository
public interface LivreRepository extends JpaRepository<Livre,
    Long> {

    // Trouver tous les livres d'un auteur
    List<Livre> findByAuteur(String auteur);

    // Trouver un livre par son ISBN
    Optional<Livre> findByIsbn(String isbn);

    // Trouver tous les livres disponibles
    List<Livre> findByDisponibleTrue();

    // Trouver les livres dont le titre contient un mot-cle
    List<Livre> findByTitreContainingIgnoreCase(String titre);

    // Trouver les livres d'un auteur qui sont disponibles
    List<Livre> findByAuteurAndDisponibleTrue(String auteur);
}
```

Listing 30: repository/LivreRepository.java - Solution complète

Solution Exercice 5 : Contrôleur REST

```java
package com.bibliotheque.livresapi.controller;

import com.bibliotheque.livresapi.model.Livre;
import com.bibliotheque.livresapi.repository.LivreRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/livres")
public class LivreController {

    @Autowired
    private LivreRepository livreRepository;

    // GET /api/livres - Recuperer tous les livres
    @GetMapping
    public List<Livre> getAllLivres() {
        return livreRepository.findAll();
    }

    // GET /api/livres/{id} - Recuperer un livre par ID
    @GetMapping("/{id}")
    public ResponseEntity<Livre> getLivreById(@PathVariable
       Long id) {
        return livreRepository.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    // POST /api/livres - Creer un nouveau livre
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Livre createLivre(@RequestBody Livre livre) {
        return livreRepository.save(livre);
    }

    // PUT /api/livres/{id} - Modifier un livre existant
    @PutMapping("/{id}")
    public ResponseEntity<Livre> updateLivre(
            @PathVariable Long id,
            @RequestBody Livre livreDetails) {

        return livreRepository.findById(id)
                .map(livre -> {
                    livre.setTitre(livreDetails.getTitre());
```

```
49                      livre.setAuteur(livreDetails.getAuteur());
50                      livre.setIsbn(livreDetails.getIsbn());
51                      livre.setAnneePublication(livreDetails.getAnneePublicat
52                      livre.setDisponible(livreDetails.isDisponible());
53                      return
                            ResponseEntity.ok(livreRepository.save(livre));
54                  })
55                  .orElse(ResponseEntity.notFound().build());
56      }
57
58      // DELETE /api/livres/{id} - Supprimer un livre
59      @DeleteMapping("/{id}")
60      public ResponseEntity<Void> deleteLivre(@PathVariable Long
            id) {
61          if (livreRepository.existsById(id)) {
62              livreRepository.deleteById(id);
63              return ResponseEntity.noContent().build();
64          }
65          return ResponseEntity.notFound().build();
66      }
67  }
```

Listing 31: controller/LivreController.java - Solution complète

---

Fin du TD1