

# Ateliers Les APIs RestFul

Pré-requis :

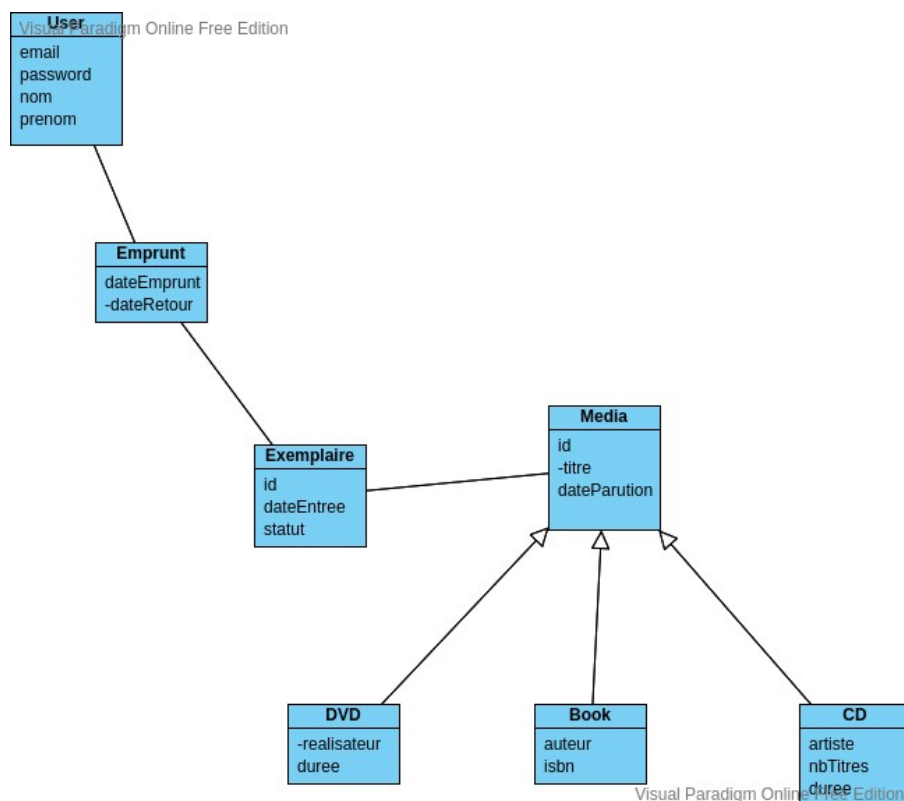
- JDK 11+
- IDE (IntelliJIDEA, Eclipse, STS, VSCode)
- Librairie lombok : <https://projectlombok.org/downloads/lombok.jar>
- Docker
- Git

## Atelier 1 : Spécification d'API

L'API visée permettra d'interagir avec une médiathèque.

La médiathèque gère un stock de DVD, CD et Livres et doit permettre à des utilisateurs d'effectuer des emprunts.

Le modèle de donnée sous-jacent est décrit via le diagramme de classe suivant :



L'API REST doit permettre les interactions suivantes :

- Sur la ressource Media :
  - Visualiser le fond documentaire et leur disponibilité à la réservation
    - Pouvoir filtrer par type de média
    - Pouvoir paginer et trier
    - Pouvoir saisir un mot-clé
  - CRUD sur Media
  - Voir les Exemplaires d'un media particulier
  - Ajouter/Supprimer/Modifier un exemplaire d'un media particulier
- Sur la ressource emprunt
  - Effectuer un emprunt pour un utilisateur
  - Restituer un emprunt pour un utilisateur
  - Visualiser les emprunts en cours d'un utilisateur

L'API devra respecter quelques règles métier :

- Un utilisateur ne peut pas avoir plus de 3 emprunts simultanés
- La date limite de restitution d'un item est d'1 semaine

Installer *swagger-editor* pour définir la spécification

Si vous avez Docker installé :

*docker run -d -p 80:8080 swaggerapi/swagger-editor*

## Atelier 2 : Génération de code à partir de OpenAPI

Nécessite npm

### 2.1 Back-end Java/Spring

Se créer un répertoire projet *mediatheque*

Installer le générateur dans le répertoire du projet :

```
npm install @openapitools/openapi-generator-cli -D
```

Générer les classes backend avec la commande suivante :

```
npx @openapitools/openapi-generator-cli generate -i ../../1_Specification/mediatheque.yaml -g spring -o . --additional-properties=delegatePattern=true --additional-properties=basePackage=org.mediatheque
```

Importer le projet Maven dans un IDE et visualiser les classes générées

### 2.2 Front-end Angular

Cette partie est inspirée de :

<https://www.kevinboosten.dev/how-i-use-an-openapi-spec-in-my-angular-projects>

Installer le framework Angular cli version 8.2.2 avec la commande suivante

```
npm install -g @angular/cli@8.2.2
```

Reprendre le projet angular fourni *mediatheque-front* et le placer au même niveau que *mediatheque*

Installer ensuite le générateur de code dans le répertoire *mediatheque-front* :

```
npm i @openapitools/openapi-generator-cli -D
```

Visualiser la commande de génération dans *package.json* et générer le code

Démarrer le serveur de développement avec :

```
ng serve
```

Accéder à l'application sur *http://localhost:4200*

### 2.3 Mock d'API

Dans le répertoire *simulator*, visualiser la référence à la spécification dans *index.ts*

Démarrer le serveur

***npm start***

Accéder à <http://localhost:9000/media>

## Atelier 3 : Consumer Driven Contract avec Spring Cloud Contract

(Adapté de <https://www.baeldung.com/spring-cloud-contract>)

### Côté producteur

Ajouter le starter *Contract Verifier* au projet **mediatheque**

Utiliser la classe de base de test fournie la placer src/test/java/org/mediatheque

Reprendre le contrat fourni et le mettre dans src/test/resources/contracts

Mettre à jour le *pom.xml* pour inclure le plugin de SpringCloudContract :

```
<!-- Generate test classes for SC Contract -->
<plugin>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-contract-maven-plugin</artifactId>
  <version>2.1.1.RELEASE</version>
  <extensions>true</extensions>
  <configuration>
    <baseClassForTests>
      org.formation.BaseTestClass
    </baseClassForTests>
  </configuration>
</plugin>
```

Générer les classes de test via ***./mvnw test-compile*** par exemple

Visualiser la classe de test puis exécuter les tests.

## Atelier 4 : Logique métier

La modélisation des classes entités et des classes d'accès aux données est fournie.

Elle identifie 2 agrégats :

- Media qui comprend les exemplaires associés
- User qui comprend les emprunts associés.

Reprendre le code fourni, se familiariser avec et vérifier que les tests des classes Repository passent

Nous voulons maintenant implémenter le service métier permettant à un utilisateur d'effectuer un emprunt.

La signature de la classe Service à respecter est la suivante :

```
public Emprunt doEmprunt(Integer idUser, List<Media> medias) throws BusinessException
```

Essayer d'appliquer le DomainModel Pattern en encapsulant les règles métier relatives à un emprunt dans la classe UserEntity

Vous pouvez tester votre implémentation en exécutant le test  
***src/test/java/org.mediatheque.UserServiceTest***

## **Atelier 5 : Couche contrôleur**

Récupérer les classes de tests fournies.

Les comprendre

### **5.1 Classes contrôleurs**

Implémenter les classes contrôleurs :

- Qui permettent de faire passer les tests lorsque les requêtes sont correctes
- De faire passer le test de Spring Cloud Contract

### **5.2 Gestion des Exceptions**

Implémenter un bean *@ControllerAdvice* de telle sorte que tous les tests passent.