Mise en place d'une API Rest avec Spring

1 Documenter votre API avec Swagger

⏩ Travail à faire :

- Suivre cette documentation afin de comprendre la mise en place d'une documentation automatique de votre API.
- Utiliser l'annotation `@Parameter` présentée dans la documentation précédente pour donner des informations sur les paramètres ou sur le corps des requêtes.

2 Utiliser ModelMapper

⏩ Travail à faire :

- Consultez ce site, ajoutez la dépendance et vérifiez l'exemple dans un test unitaire.
- Faites en sorte de définir une objet `MovieDTO` dans lequel vous allez placer des directives de validation différente (sur l'année par exemple). Vous pouvez maintenant réclamer en entrée des requêtes `POST /movies` la version DTO et ainsi mieux contrôler les contraintes qui portent sur les instances créées.
- Ajoutez à votre DTO la méthode ci-dessous et faites en sorte de renvoyer des instances de `MovieDTO` pour la requête `GET /movies`. Vous venez, facilement, de changer la forme des données renvoyées.

```
public String getCompleteName() {
    return getName() + "␣" + getYear();
}
```

3 Une courte introduction à Spring Rest Data

Pour la mise en oeuvre de **Spring Rest Data** n'oubliez pas de créer un package `myboot.app4.dao` pour les entités et les dépôts.

⏩ Travail à faire :

- Prenez soin d'ajouter le paramtere `spring.data.rest.basePath=/apibis` dans le fichier `application.properties` afin de na pas avoir de confusion entre l'API générée et les anciennes faites à la main.
- Suivez dans un premier temps cette présentation.

⏩ Travail à faire :

- Suivez dans un deuxième temps cette présentation sur le traitement des relations.
- Dans un troisième temps, aidez-vous de cette documentation pour
  - ▷ Ajouter une logique métier à l'API,
  - ▷ Utiliser le mécanisme de validation.

4 Les JSON Web Token (JWT)

Note : Nous allons, dans un sujet suivant, utiliser les JWT pour l'authentification de notre API-Rest. Vous devez, dans un premier temps, comprendre la structure d'un JWT, sa création et sa vérification.Un JWT est une structure composée de trois parties. Lisez la description de la page Wikipedia.

Travail à faire :

- Jouez avec le site `https://jwt.io/` pour construire et valider des JWT.
- AJoutez la dépendance ci-dessous pour utiliser la librairie Java JWT

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.12.6</version>
</dependency>
```

- Le code ci-dessous va créer un JWT d'une durée de vie de cinq secondes (pour tester) qui regroupe deux informations dans la charge utile. Créez un test unitaire pour exécuter ce code.

Encodage d'un JWT

```
// Maintenant et cinq secondes plus tard
Date now = new Date();
Calendar c = Calendar.getInstance();
c.setTime(now);
c.add(Calendar.SECOND, 5);
Date nowPlus5Seconds = c.getTime();

// un secret pour signer le token
String secretText = "Yn2kjibddFAWtnPJ2AFlL8WXmohJMCvigQggaEypa5E=";
byte[] secretBytes = secretText.getBytes(StandardCharsets.UTF_8);
var secret = Keys.hmacShaKeyFor(secretBytes);

// construction d'un JWT
String jws = Jwts.builder()//
        .issuer("Jean-Luc MASSAT")//
        .subject("Test JWT")//
        .claim("name", "JLM")//
        .claim("scope", "admins")//
        .issuedAt(now)//
        .expiration(nowPlus5Seconds)//
        .signWith(secret)//
        .compact();
```

- Vérifiez que le code fabriqué est bien décodé par le site `https://jwt.io/`.

⏩ **Travail à faire :** Le code ci-dessous assure le décodage d'un jeton JWT. Vous pouvez explorer `jwsDecoded` pour récupérer les informations.

Décodage d'un JWT

```
Jws<Claims> jwsDecoded = Jwts.parser()//
        .verifyWith(secret)//
        .build()//
        .parseSignedClaims(jws);
```

Vérifiez, par un test unitaire, que le jeton n'est plus utilisable après cinq secondes.