Projet MI7
API REST de la classification des livres

Quentin Carrel - Evan Galleron
Camille Bonacorsi - Loïc Delon

Introduction

Le but de ce projet consiste en la réalisation d'un API REST.
Le paradigme REST repose sur quelques principes clefs:

- Une interface uniforme, permettant d'accéder à toutes les ressources de la même manière, et au travers de laquelle chaque "chemin" (URI) mène toujours à la même ressource
- Une séparation "client-serveur", stipulant que la manière d'accéder à des données pour les utilisateurs ne doit pas être directement dépendante de la manière dont ces données sont représentées en interne, côté serveur
- L'absence d'état, permettant à chaque requête de ne pas dépendre d'une quelconque manipulation préalable, et de s'autosuffire
- La "chacheabilité", préconisant la présentation de ressources pouvant être mises en cache, ainsi qu'une indication de si une ressource spécifique est adaptée à la mise en cache ou non (pas forcément le cas pour des données qui évolueraient rapidement, par exemple)
- Une architecture à plusieurs couches, précisant qu'il est possible voire probable que les requêtes ne s'effectuent pas directement entre le client et la source de données. Ces multiples couches doivent être transparentes pour l'utilisateur du client.

Pour ce projet, nous avons choisi de représenter le lien entre un livre, ses éditions, son éditeur et son auteur à l'aide d'un diagramme UML, d'une spécification OpenAPI, d'une base de données SQL et d'un serveur Node.js.

Choix de l'API

L'API que développée permet de consulter des informations concernant des livres. La structuration de l'API rend plus particulièrement facile l'accès aux différents ouvrages écrits par un auteur, ou publiés par une maison d'édition.
La modélisation se base sur les constats suivants:

1. Un auteur écrit des livres
2. Un livre a été écrit lors d'une certaine décennie
3. Un livre appartient à un certain genre
4. Une maison d'édition publie des éditions dans des collections
5. Un livre peut ne pas avoir été publié

Ces constats sont représentés dans ce diagramme de classe :

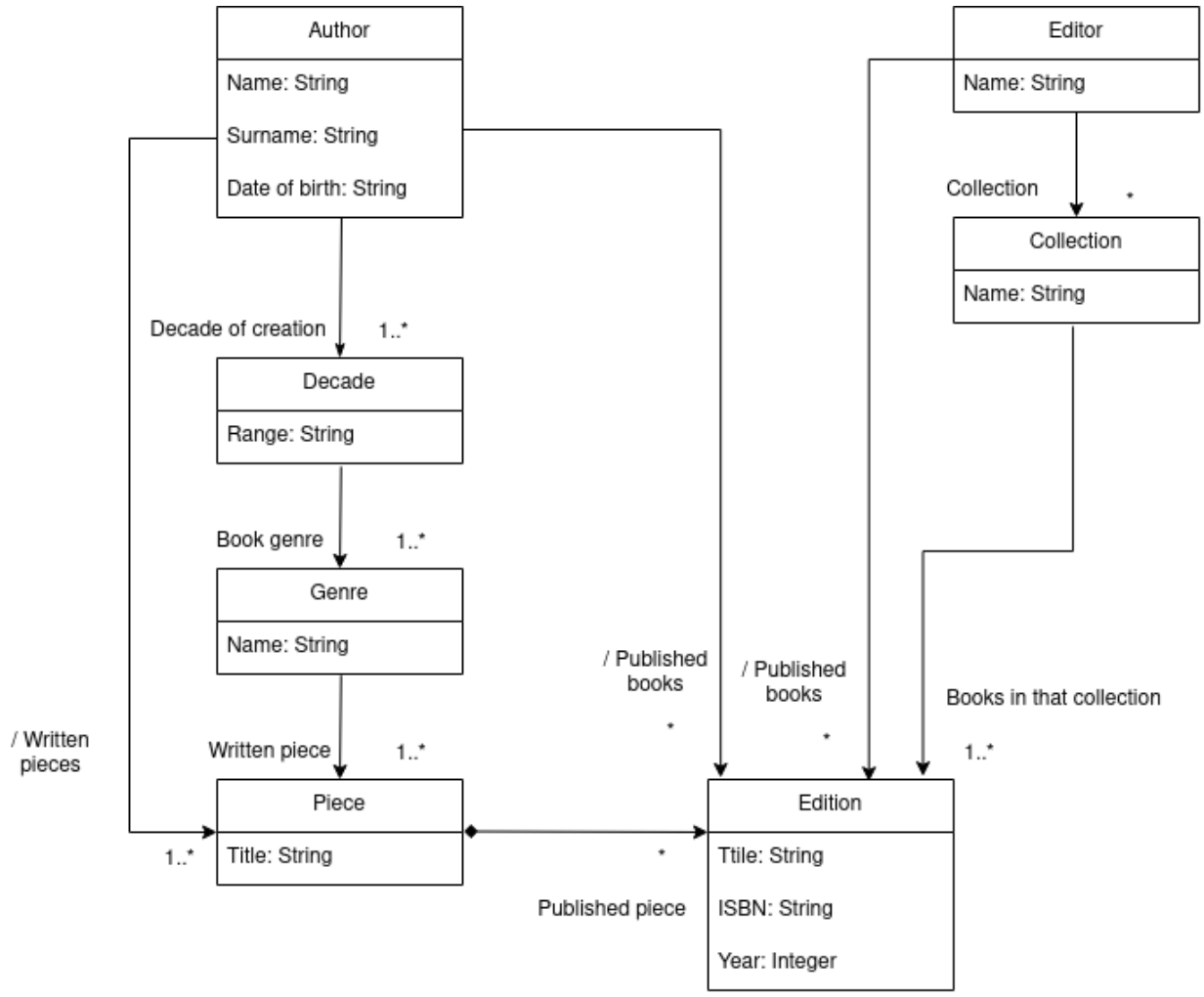

Diagramme UML des données de l'API

Les attributs dérivés "*Written pieces*" et "*Published books*" permettent d'accéder directement à toutes les oeuvres écrites et publiées par un auteur ou une maison d'édition.

La création de ce diagramme de classe permet assez facilement la réalisation d'une spécification OpenAPI, notamment grâce à l'outil Swagger. La spécification entière est accessible dans le fichier "openapi_uit.yaml" à la racine du projet

Voici un extrait de cette spécification :

```yaml
        type: string
    links:
      properties:
        href:
          type: string
        method:
          type: string

paths:
  /:
    get:
      summary: Entry points
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/links'
  /authors:
    get:
      summary: List of authors
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/author'
        '400':
          description: No authors found
    put:
      summary: Add/Overwrite an author
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/author'
      responses:
```

Extrait de la spécification OpenAPI

Swagger a également été choisi pour la fonctionnalité proposée de générer des *stubs* dans de multiples langages à partir d'une spécification OpenAPI. Ce choix est motivé par le gain de temps associé, et en raison d'une expérience préalable avec l'outil.
La première itération du serveur a donc été générée avec la spécification construite à partir du diagramme UML réalisé au préalable, mais l'évolution du serveur, du diagramme UML et de la spécification OpenAPI a par la suite suivi un cours organique, s'adaptant aux besoins et contraintes émergeant du développement du serveur.
Toujours pour des raisons de familiarité avec les outils, "nodejs-server" a été utilisé comme base de serveur.
Bien qu'un backend en javascript n'offre pas la robustesse d'un langage typé ou les performances d'un langage compilé, sa facilité d'utilisation et la nature d'exercice de la tâche à accomplir en font un choix adapté.

Les données sont enregistrées dans une base SQL, et modifiées/lues à l'aide de la librairie SQLite. Dans le cas d'une application devant supporter beaucoup d'écritures concurrentes ou d'une séparation forte de la BDD et de l'application, il serait envisageable d'utiliser un DBMS plus riche comme PostgreSQL.

Pour des questions de temps, peu de fonctionnalités ont été développées. Ce serveur REST permet de:

- Parcourir les données (HATEOAS)
- Ajouter un auteur (endpoint POST/PUT)
- Supprimer un auteur (endpoint DELETE)

À l'exception de quelques oeuvres, du nom des collections et de deux auteurs, toutes les données ont été générées aléatoirement, à partir d'une liste de noms et prénoms pour les auteurs, de générateur de nom d'entreprise pour les éditeurs (source perdue), et d'un outil spécialisé pour les noms des livres.
La nature factice des données utilisées et l'utilisation d'une modèle SQL a demandé une modification de la manière dont les données sont enregistrées et surtout liées entre elles. Afin d'obtenir un système cohérent et fonctionnel le plus rapidement possible, toutes les compositions ont été supprimées et chaque lien fait contient l'ensemble des entités.
Toutes les données et leur représentation sont disponibles dans le dossier "sql" du projet.

Utilisation

La mise en place du serveur requiert nodejs. Il suffit de dézipper le projet, se placer à sa racine, exécuter `npm install` puis `npm start`.
Après ceci, l'API est accessible sur le port 8080 de l'adresse localhost (adresse http://localhost:8080 dans un navigateur).
L'accès à l'API se fait au travers d'un outil permettant d'envoyer des requêtes HTTP (navigateur, curl, wget, …).
Une documentation de l'API générée automatiquement est disponible à l'url http://localhost:8080/docs.
La navigation est guidée par des suggestions de liens associés aux attributs "href" des objets figurant dans les réponses reçues.

Conclusion

Ce projet avait pour objectif de nous faire découvrir les différentes facettes de la création d'une API REST, à savoir la modélisation des données, la rédaction d'une spécification, et le développement d'un serveur. La réalisation de cette tâche s'est effectuée en 6 étapes.
La recherche d'informations à présenter en premier lieu, suivie d'une modélisation de ces données à l'aide d'un diagramme de classe dans un second temps.
L'écriture de la spécification à partir de ce diagramme dans un troisième temps.
La quatrième étape consistant en la génération du serveur à partir de cette spécification, suivie du choix et de l'exercice de la création de données factices afin de gagner du temps en

cinquième étape. Enfin, des cycles de développement et révision des étapes précédentes au fur et à mesure que les besoins réels du projet émergeaient.

L'utilisation de données générées n'ayant aucun lien entre elles, couplé à l'utilisation une base de données SQL, a engendré des difficultés conduisant à des pratiques peu orthodoxes (une unique table de liaison avec une colonne par entité).
Cependant, ceci a ultimement permis d'en apprendre plus sur la navigabilité des données liées à plusieurs éléments, et les conditions requises pour pouvoir parcourir ces données de manière "déterministe" (si chaque entité n'était reliée directement qu'à une seul autre entité, chaque genre de livre devrait par exemple avoir une instance propre à chaque auteur).

L'utilisation de stubs générés automatiquement et utilisant une bibliothèque peu documentée (oas3-Tools), bien qu'une solution permettant un développement plus rapide, a abouti à l'écriture d'un code qui n'est pas entièrement maîtrisé. Cela se manifeste notamment par la réception de code 200 par le client, même lorsque la spécification indique qu'un code 400 ou 404 devrait être envoyé/reçu, aucune information quant à la gestion des codes de statut HTTP avec cette bibliothèque logicielle n'ayant été trouvée.