TD OAS : OpenAPI Specification

Consignes :

- Créer un repo sur Github et push en utilisant des commits selon les standards des conventional commit (https://www.conventionalcommits.org/en/v1.0.0/) tout au long de vos modifications.
- L'intégralité du document doit être écrit en ANGLAIS.

TD1 : Rappel spécification OpenAPI 3.1.0

Thème : Gestion de cours étudiants

Nous voulons documenter la spécification d'une gestion des cours des étudiants. Concentrons nous dans ce premier TD sur la spécification de la gestion des groupes et des étudiants.

1. Au niveau de l'OpenAPI, voici le travail attendu :
    a. Points d'entrée attendus :
    - /groups avec les méthodes attendues GET, POST, PUT, DELETE
        - Dans GET /groups, il est possible d'effectuer un filtre par groupName et par groupYear (une intervalle de dates)
        - PUT /groups est une requête idempotente.

    - /students avec les méthodes attendues GET, POST, PUT, DELETE
        - Dans GET /students, il est possible d'effectuer un filtre par studentName
        - PUT /students est une requête idempotente
        - Dans la v1.0.0 de l'API, le type de group peut juste être de type texte. Pour faire référence notamment au nom du groupe : J2 par exemple

    b. Caractéristiques des ressources :
    - Group : ID, groupName, groupYear, promotion (valeurs possibles G, H, J, K), studentNb, students* (liste de Student)

    - Student* : ID, Name, Sex (val*eurs possibles : M ou F), birthdate, reference (ex: STD23XXX), group (type texte suffit pour la version 1.0.0)

2. À travers cette spécification, nous voulons tester sur Postman si la spécification répond vraiment à nos attentes. "Uploader" la spécification sur SwaggerHub afin de bénéficier du service du serveur mock, et faites le test de chaque point d'entrée sur Postman en créant une collection Postman avec toutes les requêtes.

Tips : il existe une fonctionnalité sur Postman, qui permet d'uploader une spécification OpenAPI (fichier .yaml) qui génère directement la collection Postman associée.
https://github.com/hei-prog-3-backend/library-management?fbclid=IwAR0kwJqtvYEdzfwaOtITKUT3vqpS2v9He6pJQaUk2liVI_BAX7TyEqhpXyo

Points d'entrée :

- /books : GET, PUT, DELETE
  - Dans GET /books, il est possible d'effectuer un filtre par bookName et par releaseDate (une intervalle de dates)
  - PUT /books est une requête idempotente. Dans la v1.0.0, la propriété author est un string.
- /authors: GET, PUT, DELETE
  - Dans GET /authors, il est possible d'effectuer un filtre par authorName
  - PUT /authors est une requête idempotente

Caractéristiques des ressources :

- Book : ID, bookName, author*, pageNumbers, topic (valeurs possibles : ROMANCE, COMEDY, OTHER), releaseDate
- Author* : ID, Name, Sex (valeurs possibles : M ou F)

TD2 : Bonne pratique sur la spécification OpenAPI 3.0.3

Thème : Bibliothèque
Pré-requis : les 3 TODO de la TD1 finies.
Consignes :

- Renommer votre branche actuelle sur Git **oas-td1-stdxxx** où vous allez remplacer **stdxxx** par votre référence étudiant, ensuite changez dans une branche **oas-td2-stdxxx** pour la suite de l'exercice
- Mettez à jour la version de votre spécification API. (De 1.0.0 vers x.x.x)

1. Maintenant que Book est associé à Author, lors de notre requête PUT /books, nous devons obligatoirement fournir un objet Author au moment de la création d'un Book. Notre objectif sera de rendre la création d'un Book indépendant de la création d'un Author. Autrement dit, nous n'aurons plus besoin de fournir un Author au moment de la création d'un Book, mais en retour, nous obtiendrons toujours la propriété Author qui sera null pour le moment.

   Pour cela, il faut que vous créez un composant objet intitulé CrupdateBook, qui va être caractérisé par les attributs suivants uniquement : ID, bookName, pageNumbers, topic (valeurs possibles : ROMANCE, COMEDY, OTHER), releaseDate.

   Le composant Book, qui va être retournée en tant que contenu de la réponse 200, ne

doit pas changer mais devrait toujours avoir les propriétés bookName, pageNumbers, topic (valeurs possibles : ROMANCE, COMEDY, OTHER), releaseDate. Pourtant, nous avons évoqué qu'une redondance (répétition ou copier-coller) est très déprécié, utilisez l'**héritage** (mot clé **allOf**) et consulter la documentation https://spec.openapis.org/oas/latest.html pour solutionner ce problème.

2. Une fois la question précédente effectuée, il n'est plus possible pour nous d'attacher un Author à un Book. Il faut créer les deux nouveaux points d'entrée suivants qui permettent d'effectuer cette action :
    a. PUT /books/{bookId}/authors/{authorId} sans corps de requête, qui permet de modifier l'author d'un Book en spécifiant chacun leurs identifiants (ID) respectifs.

    b. PUT /books/authors avec un corps de requête qui prend en paramètre une liste d'objet intitulé UpdateBookAuthor, caractérisé les propriétés par bookId et authorId.

        i. Pourquoi UpdateBookAuthor possède uniquement l'identifiant de CrupdateBook et l'identifiant de Author, mais sans les autres propriétés telles que bookName et authorName comme dans leur composant respectif ?

        ii. Dans quel cas, UpdateBookAuthor devrait avoir les propriétés de CrupdateBook et de Author ?

3. Pour GET /books, ajoutez des paramètres de requêtes ou **query parameters** pour effectuer de la pagination (page et pageSize). Ces paramètres ne sont pas requis, et par défaut, indiquez dans la spécification que la valeur par défaut de page est 1 si aucune n'est fournie et la valeur par défaut de pageSize est 50 si aucune n'est fournie.
    a. Pourquoi les paginations sont-elles nécessaires ?

4. Pour GET /authors, ajoutez des paramètres d'URL ou **path parameters** pour effectuer de la pagination (page et pageSize). Ces paramètres ne sont pas requis, et par défaut, indiquez dans la spécification que la valeur par défaut de page est 1 si aucune n'est fournie et la valeur par défaut de pageSize est 50 si aucune n'est fournie.
    a. Est ce qu'on peut gérer la pagination à travers les entêtes de la requête ? Justifiez votre réponse.
    b. Est ce qu'on doit gérer la pagination à travers les entêtes de la requête ? Justifiez votre réponse.

TD3 : Suite bonne pratique sur la spécification OpenAPI 3.0.3

Thème : Bibliothèque
Pré-requis : TD2 doit être finie.
Consignes :

- Si ce n'est pas encore fait, renommez votre branche actuelle sur Git **oas-td2-stdxxx** où vous allez remplacer **stdxxx** par votre référence étudiant, ensuite changez dans une branche **oas-td3-stdxxx** pour la suite de l'exercice.
- Mettez à jour la version de votre spécification API. (De 1.0.0 vers x.x.x)
- Pour tous les autres TD à suivre, n'oubliez pas de suivre cette procédure.

1. Si ce n'est pas encore fait, créez des composants réutilisables dans components/parameters mais **PAS** dans components/schemas, pour gérer la pagination à travers des query parameters que nous allons intitulé "queryPagination" et path parameters que nous allons intitulé "pathPagination". Remplacez les paramètres qui ont géré la pagination plus tôt (dans GET /books et GET /authors) avec ces composants.

2. Remarquez que pour le moment, nous n'avons toujours qu'une réponse 200 retournée par notre serveur pour toutes les opérations. Pourtant, il est tout à fait possible que le serveur nous retourne d'autres réponses, notamment des réponses 3xx, 4xx ou 5xx. Pour le moment, supposons que le serveur ne peut retourner uniquement que les réponses 200, 400, 403 et 500.
    a. Chaque réponse issue du serveur possède un même structure par défaut, notamment il est caractérisé par :
        i. un objet intitulé "status" composé des propriétés : code de statut et message.
        ii. un objet intitulé "body" contenant les valeurs des objets retournés

    Autrement dit, par défaut, que ce soit une réponse 200, 400, 403 ou 500, il faut qu'il suit cette structure. La méthode la plus simple serait donc de créer un composant un à un pour chacun de ces objets de réponses, mais il y aurait encore beaucoup de redondance si vous le faites pour toutes les réponses. Pour éviter ces redondances, vous pouvez exploiter encore une fois **l'héritage** et effectuer les modifications nécessaires pour pouvoir adapter correctement sans que pour autant notre spécification soit totalement abstraite, en particulier pour les réponses 200.

3. Notre bibliothèque a évolué car 20.000 livres viennent d'arriver venant de 2 fournisseurs différents, soit 10.000 chacun. Le premier fournisseur nous a livrés deux fichiers Excel (au format .xlsx) : l'un contenant les informations des auteurs (authorName, sex) et l'autre contenant les informations des livres (bookName, author, pageNumbers, topic, releaseDate, idAuthor). En particulier, les données sont déjà associées à un idAuthor pour permettre d'associer directement les livres à un auteur au moment de l'importation.

Le second fournisseur a également fait de même, à l'exception qu'ils nous a livrés des fichiers au format JSON à la place des fichiers Excel, contenant une liste d'objet JSON ayant les mêmes caractéristiques (bookName, author, pageNumbers, topic, releaseDate, idAuthor).
Ces fichiers ont pour but de faciliter l'import des données de ces nouveaux livres et d'auteurs dans notre système.

a. Notre premier constat c'est que les fichiers (Excel ou JSON) qu'ils nous ont fourni ne possèdent pas d'ID, il faut ainsi que ça soit le serveur qui gère la création d'ID lors des opérations d'import. On propose donc de créer deux points d'entrées dorénavant : POST /books/import et POST /authors/import. Respectivement, POST /books/import doit impérativement retourner la liste des livres importés, soit une liste de Book, et pareil pour POST /authors/import avec une liste de Author.

b. Le deuxième constat c'est maintenant qu'il y a deux types de données possibles en entrée mais un même type de donnée en sortie pour l'importation d'auteurs et de livres. Pour rappel, pour indiquer le type d'un fichier sur OAS, il faut utiliser le type MIME, par exemple "application/json" indique que le type du fichier attendu est un objet JSON. Il existe également un type MIME pour les fichiers Excel. Maintenant dans la spécification de ce choix, voici les possibilités :
   i. Pour l'opération d'importation d'auteurs : soit un fichier Excel, soit un fichier JSON avec une liste d'objets JSON attendue, qu'on va intituler "ImportAuthor" caractérisé par authorName, sex.
   ii. Pour l'opération d'importation de livres : soit un fichier Excel, soit un fichier JSON avec un liste d'objets JSON attendues, qu'on va intituler "ImportBook" caractérisé par bookName, author, pageNumbers, topic, releaseDate, idAuthor.

   Pour gérer ce choix, il y a plusieurs manières de le faire. Pour ce TD, nous attendons que vous utilisiez le polymorphisme (mot clé **"oneOf"**).

TD4 : Modéliser un cahier de charges

Thème : Bibliothèque
Pré-requis : TD plus tôt doit être fini.
Consigne à revoir (nouvelle branche, etc) et travail en binôme du même groupe.
Maintenant que vous êtes un peu plus à l'aise avec la spécification, vous devez être capable de modéliser un cahier des charges selon les spécifications OAS.
Notes importantes : Lisez les deux parties avant de commencer le TD.

Pour le moment, notre bibliothèque n'a qu'une notion de livres et d'auteurs, mais dans une bibliothèque réelle, il y a des visiteurs qui empruntent et qui rendent des livres, et des administrateurs qui font le suivi de ces livres empruntés.

Partie 1 : Gestion des visiteurs qui empruntent et rendent des livres

1. Intégrer les visiteurs (ID, name, reference) dans notre API.
2. Modéliser les actions (emprunter et rendre des livres) faites par les visiteurs. Autrement dit, effectuer les ajouts ou modifications nécessaires à la version actuelle de notre API pour pouvoir gérer cela.
3. Après la question 2, pour chaque livre, il devrait être possible de savoir si celui-ci est disponible ou encore emprunté par une personne. Hormis la modélisation de l'emprunt et du rendu, modifier l'API pour pouvoir obtenir la liste des livres empruntés et des livres disponibles.
4. Après la question 2, pour chaque visiteur, l'administrateur devrait pouvoir savoir quels livres ont été empruntés et rendus par ce visiteur.
    a. Modifier l'API en conséquence.
    b. En particulier, il veut pouvoir connaître les activités de ce visiteur pendant une période donnée, par exemple, pendant 1 mois donné, pendant 1 semaine donnée ou même entre une intervalle d'heure bien précis à une date donnée. Par activités, on fait bien référence aux emprunts et rendus de livres que ce visiteur a effectué et quels sont les livres qui sont concernés s'il y en a eu.

Partie 2 : Gestion de la permission et de la sécurité

Actuellement, d'après notre API, tous les points d'entrées peuvent être accessibles par n'importe qui et cela montre une faille de sécurité majeure dans notre système. Voilà comment devrait être géré les permissions :

- Tout le monde (administrateur et visiteurs, autrement dit les personnes non identifiées ou non authentifiées) peut consulter la liste des livres disponibles dans la bibliothèque ainsi que des auteurs, et effectuer des recherches là-dessus.

- Seuls les administrateurs peuvent emprunter ou rendre un livre. Autrement dit, lorsque les visiteurs veulent emprunter un livre, ils doivent se rapprocher d'un administrateur et c'est celui-ci qui va effectuer l'action d'emprunter ou de rendre un livre dans le système, au nom du visiteur.

- Seuls les administrateurs peuvent ajouter ou importer un nouvel auteur ou un nouveau livre.

Dans ce TD, supposons que nous allons utiliser essentiellement le protocole d'autorisation OAuth 2.0 pour gérer les permissions.
Pour connaître les syntaxes qui spécifient ces autorisations et la sécurité en générale, consultez la documentation officielle : https://spec.openapis.org/oas/latest.html