ProgRes - Projet - Web Api

Le but du projet est d'utiliser la librairie Python `bottle`, pour réimplémenter l'API du site dblp qui regroupe l'ensemble des publications en informatique, avec quelques fonctionnalités supplémentaires. Le site dblp met en ligne l'ensemble des publications sous forme d'un fichier Xml. Vous devez donc le télécharger et utiliser les données qu'il contient afin de creer votre API. Dans la suite, on appelle *publication*, un élément de type `article`, `inproceedings`, `proceedings`, `book`, `incollection`, `phdthesis`, ou `mastersthesis`.

Le fichier xml (se trouvant à l'adresse `http://dblp.uni-trier.de/xml/`) doit être parsé pour récupérer chaque publication (quelque soit son type). Uniquement les champs `author`, `title`, `year`, `journal`, et `booktitle` seront récupéré (on ignorera les autres champs). On pourra remarquer que soit le champs `journal`, soit le champs `booktitle` est défini par publication (`booktitle` correspond en fait au nom de la conférence dans laquelle est publié l'article). On ignorera le type des publications, `article`, `inproceedings`, etc, quelque soit leur type, elles seront considérées comme des *publications*.

En raison de la quantité limitée de mémoire RAM disponnible sur certains ordinateurs, vous êtes autorisés à ne considérer que les publications des 5 dernières années (voir moins, mais ce parmètre doit pouvoir être modifié dans le code source).

Votre API doit accepter les routes suivantes :

- `/publications/{id}` : avec `id` l'identifiant d'une publication (à vous de choisir quels sont les identifiants), qui retourne la publication correspondante.
- `/publications` : qui retourne les 100 premières publications (la valeurs 100 pourra être modifiée par la suite avec un paramètre d'url `limit`)
- `/authors/{name}` : avec `name` le nom d'un auteur, qui retourne les informations concernant un auteur (nombres de publications, nombre de co-auteurs).
- `/authors/{name}/publications` : avec `name` le nom d'un auteur, liste les publications d'un auteur.
- `/authors/{name}/coauthors` : avec `name` le nom d'un auteur, liste les co-auteurs d'un auteur.
- `/search/authors/{searchString}` : avec `searchString` une chaine de charactères permettant de chercher un auteur. Cette route retourne la liste des auteurs dont le nom contient `searchString` (ex: /search/authors/w retourne la liste de tous les autheurs contenant un `w` ou un `W`).
- `/search/publications/{searchString}` : avec `searchString` une chaine de charactères et retourne la liste des publications dont le titre contient `searchString`. La route accepte un paramètre d'url `filter` de la forme `key1:value1,key2:value2,...` afin d'affiner la recherche aux publications dont la clef `keyi` contient `valuei`. Ainsi, la recherche `/search/publications/robots?filter=author:Jean,journal:acm` doit retourner la liste des publications dont le titre contient `robots`, dont l'auteur contient `Jean` et publiées dans un journal contenant `acm`.
- Les routes permettant la recherches autorisent l'utilsation du charactère `%`, respectivement `*`, peut remplacer un charactère quelconque, respectivement une suite de charactères quelconques. (ex: les requêtes `/search/authors/quen%%% br%m%s`, `/search/authors/*tin Bram*`, ou `/search/authors/Que*amas` doivent trouver l'auteur `Quentin Bramas`). De plus, les comparaisons doivent être insensible à la casse.
- `/authors/{name_origine}/distance/{name_destination}` : avec `name_origine` et `name_destination` deux auteurs, qui retourne la distance de collaboration entre les deux auteurs donnés. Cette distance est définie comme la longueur du plus petit chemin `(name_origine, auth1, auth2, ..., authX, name_destination)`, où `name_origine` et `auth1` sont co-auteurs, `auth1` et `auth2` sont co-auteurs, ... et `authX` et `name_destination` sont co-auteurs. En particulier deux co-auteurs sont à distance 1. En plus de retourner la distance, la réponse doit contenir le chemin entre les deux auteurs.

Votre Api doit présenter les charactéristiques suivantes:

- Toutes les erreurs doivent avoir le même format.
- Chaque route doit être documentée, avec le format de retour, les erreurs possibles et une explications des paramètres.
- Chaque route qui retourne une liste, doit retourner au maximum 100 éléments et accepter des paramètre d'URL `start` et `count` qui permettent d'afficher `count` éléments, à partir du `start`-ième élément. Ex: `/search/authors/*` doit retourner les 100 premiers auteurs, `/search/authors/*?start=100` affiche les 100 suivants, et `/search/authors/*?start=200&count=2` affiche les 2 éléments suivants.

- Pour chaque route qui retourne une liste, les éléments retournés doivent pouvoir être triés par rapport à un champs donné dans un paramètre d'URL `order`. Ex: `/search/publications/*?order=journal` affiche les 100 premières publications triées dans l'ordre alphabétique du nom du journal dans lequel elles aparaissent.
- Chaque route doit accepté un paramètre d'URL `fields` contenant une liste de champs séparés par des virgules `champs1,champs2,...` permettant de limiter le contenu retourné aux champs indiqués (par défaut, on retourne tous les champs).

Conseils

- Lorsque vous effectuez des tests, limitez le nombre de publications que vous parsez et que vous chargez en mémoire. Ensuite si tout fonctionne, testez votre API avec plus de publications.
- Commencez par quelques routes simples, et ajoutez les fonctionnalités une par une. Passez à une nouvelle fonctionnalité après seulement avoir proprement terminé, documenté et testé la précédente.
- Parser le XML pouvant être long, vous pouvez le parser et le convertir dans un autre format contenant uniquement les informations qui vous intéresse, afin de le charger plus rapidement lors de vos teste.