PROGRES - Mini-Projet 2
API Web

Sébastien Tixeuil - Sebastien.Tixeuil@lip6.fr

Le but de ce mini-projet est d'utiliser (entre autres) la bibliothèque Python `bottle`, pour proposer d'une part une API spécifique au site `http://dblp.uni-trier.de/db/ht/` qui regroupe l'ensemble des publications scientifiques en informatique, et d'autre part un site web qui permet d'utiliser l'API précédente.
Le site `dblp` propose l'ensemble des publications sous la forme d'un fichier Xml. Il faut donc télécharger ce fichier et utiliser les données qu'il contient afin de créer votre API. Dans la suite, on appelle *publication*, un élément de type `` `article` ``, `` `inproceedings` ``, `` `proceedings` ``, `` `book` ``, `` `incollection` ``, `` `phdthesis` ``, ou `` `mastersthesis` ``.
Le fichier Xml se trouve à l'adresse `http://dblp.uni-trier.de/xml/`. Il doit être analysé pour récupérer chaque publication (quel que soit son type). L'API demandée porte seulement sur les champs `` `author` ``, `` `title` ``, `` `year` ``, `` `journal` ``, et `` `booktitle` `` (les autres champs peuvent donc être ignorés). On peut remarquer que pour une publication, soit le champ `` `journal` ``, soit le champ `` `booktitle` `` est défini (`` `booktitle` `` correspond au nom de la conférence dans laquelle est publié l'article, `` `journal` `` correspond à la revue scientifique dans laquelle est publié l'article).
Note: En raison de la quantité limitée de mémoire RAM disponible sur certains ordinateurs, il est possible de ne considérer que les publications des 5 dernières années (voire moins, mais ce paramètre doit pouvoir être modifié dans le code source).

Exercice 1. Mise à disposition d'une API Web

Réaliser en Python et en utilisant la bibliothèque `bottle` un serveur Web qui implémente l'API Web suivante :

- `` `/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 par défaut les 100 premières publications. La valeur 100 peut être modifiée au moyen d'un paramètre d'url `` `limit` ``).
- `` `/authors/{name}` `` : avec `` `name` `` le nom d'un auteur, qui retourne des informations concernant un auteur : nombres de publications dont il est co-auteur, nombre de co-auteurs.
- `` `/authors/{name}/publications` `` : avec `` `name` `` le nom d'un auteur, qui liste les publications d'un auteur.
- `` `/authors/{name}/coauthors` `` : avec `` `name` `` le nom d'un auteur, qui liste les co-auteurs d'un auteur.
- `` `/search/authors/{searchString}` `` : avec `` `searchString` `` une chaine de caractères permettant de chercher un auteur. Cette route retourne la liste des auteurs dont le nom contient `` `searchString` `` (par exemple, `/search/authors/w` retourne la liste de tous les auteurs dont le nom contient un `` `w` `` ou un `` `W` ``).
- `` `/search/publications/{searchString}` ``: avec `` `searchString` `` une chaine de caractères et qui 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`.
- `/authors/{name_origin}/distance/{name_destination}` : avec `name_origin` et `name_destination` deux noms d'auteurs, qui retourne la distance de collaboration entre les deux auteurs nommés. Cette distance est définie comme la longueur du plus petit chemin `(name_origin, 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 plus court chemin entre les deux auteurs.

L'API ainsi développée doit présenter les caractéristiques 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. Par exemple: `/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 champ donné dans un paramètre d'URL `order`. Par exemple: `/search/publications/*?order=journal` affiche les 100 premières publications triées dans l'ordre alphabétique du nom du journal dans lequel elles apparaissent.

Exercice 2. Test unitaire d'une API Web

Réaliser en Python et en utilisant la bibliothèque `unittest` un programme qui teste le bon fonctionnement de l'API Web développée à l'exercice 1.

Exercice 3. Site Web d'utilisation d'une API Web

Réaliser en Python et en utilisant la bibliothèque `bottle` un serveur Web qui utilise l'API Web développée à l'exercice 1 pour proposer à l'utilisateur une interface Web graphique qui lui permette d'obtenir en entrant les informations pertinentes dans un formulaire Web :
- la liste *complète* des publications et la liste *complète* des coauteurs d'un auteur, possiblement triées alphabétiquement. Cet auteur peut au préalable être recherché via une sous-séquence de caractères apparaissant dans son nom.
- La distance entre deux auteurs. Ces auteurs peuvent au préalable être recherchés chacun via une sous-séquence de caractères apparaissant dans leur nom.