MU4IN014 – Mini-Projet 1 : Rendu

Sommaire :

Introduction

Le mini-projet2 a été réalisé par DIEP Richard(3700573) et comporte trois exercices. Le premier exercice correspond à la réalisation consiste à implémenter un API WEB avec la bibliothèque bottle. Le second exercice consiste à tester l'implémentation de l'exercice 1. L'exercice 3 est la réalisation d'une interface web graphique qui permet de communiquer avec l'api web générée dans l'exercice 1. Dans un premier temps, nous allons parler de l'implémentation de l'api server et ce dont il est capable de réaliser. Dans un second temps, nous verrons quelques tests effectués. Et pour finir, nous observerons la façon d'interagir avec l'interface web graphique, l'api web.

API Server

La route « /publications »

Cette route permet d'obtenir les cents premières publications du fichier local sélectionné au préalable dans le code source. Cette contrainte peut être modifiée en ajoutant en paramètre « limit=n », avec n la valeur. Avant de rechercher les publications, une vérification de l'existence et la valeur de « limit » est nécessaire car elle ne doit pas être en dessous de 1, et doit être un entier. En fonction de l'activation de la limite, la route retourne un dictionnaire contenant une liste de dictionnaires des publications récoltées. A savoir que pour toutes les routes qui retournent un dictionnaire contenant plusieurs clés similaires possibles, un indice est ajouté. Par exemple {'author' :'James', 'author0' :'Bond'}.

Une variante de cette route est « /publications/ id », avec id un entier supérieur à 0. Dans le cas où l'id est inférieur à 1, une erreur est envoyée au client. Cette route permet de récupérer la publication ayant l'id correspondant. L'association de l'identifiant est faite dans l'ordre du fichier xml, c'est-à-dire que l'id est dépendant de la façon dont les données sont organisées dans le fichier xml. La fonction gérant cette route retourne un dictionnaire contenant un dictionnaire des informations sur l'id.

La route « /authors/<name> »

Nous avons ensuite une route permettant de savoir pour « name », le nom d'un auteur, combien de coauteurs possède-t-il, mais aussi combien de fois l'auteur est un coauteur. Il faut savoir qu'on ne prend en compte que les auteurs, et donc les éditeurs par exemple ne sont pas pris en compte. Une erreur est envoyée au client si un problème est rencontré, sinon un dictionnaire contenant un dictionnaire informant le nombre de fois dont l'auteur est coauteur et le nombre de coauteurs qu'il possède.

Une autre route est possible : « /authors/<name>/publications ». Elle permet de retourner toutes les publications dont « name » est un auteur. Le nom doit être identique, c'est-à-dire qu'il ne faut pas seulement assurer que le nom soit dans le nom d'un auteur. Par exemple, si on recherche « Louise A. Dennis », on ne peut pas entrer dans la recherche « Louise A. » pour trouver l'auteur. La fonction qui gère cette route possède plusieurs paramètres : « start » pour indiquer à partir de quel indice on veut récupérer les publications, « count » pour indiquer le nombre de publications on souhaite voir sachant qu'il y a une limite de 100 et « order » pour ordonner les données.

Toutes les routes qui possèdent ces paramètres possèdent une vérification avant la recherche des informations, c'est-à-dire qu'il faut assurer que les paramètres start et count ne doivent pas être inférieur à 0 et doivent être des entiers. Si les paramètres ne sont pas respectés, une erreur est envoyée au client. Le paramètre order est aussi vérifié à la fin de la recherche en comparant la clé la chaîne de caractère dans order et les clés qui se trouvent les dictionnaires contenant les informations sur les publications. S'il y a au moins une clé de trouvée, les dictionnaires ne possédant pas ces clés sont mis à la fin de la liste et les autres sont ordonnés. La fonction gérant cette route retourne au maximum 100 valeurs sauf si count est activé, et les données retournées sont sous la forme d'un dictionnaire contenant une liste de dictionnaire.

Enfin, la dernière route correspondant à l'auteur est « /authors/<name>/coauthors ». La fonction pour cette route est essentiellement une fusion de la fonction pour compter le nombre de coauteurs et la fonction ci-dessus. En effet, celle qui permet de compter le nombre de coauteur est en réalité une liste contenant de dictionnaires de tous les coauteurs. Ce qui signifie qu'on peut retourner cette liste directement au lieu de compter le nombre de coauteurs. Ensuite, on y ajoute les paramètres « start », « count » et « order » avec toutes les vérifications au début comme la fonction ci-dessus. Cependant, order ne vérifie que lorsqu'on souhaite ordonner par « author » car il n'y a que cette clé de disponible dans le dictionnaire. Le résultat retourné est un dictionnaire contenant une liste de dictionnaires, sachant qu'au maximum, il n'y a que 100 résultats retournés.

La route « /search »

La première route « /search/authors/<searchString> » permet de rechercher tous les auteurs qui contiennent la chaîne de caractère searchString. La fonction qui gère cette route retourne aussi au maximum 100 résultats et possède les paramètres « start », « count » et « order ». Ici, l'ordre est géré de la même façon que pour la route précédente puisqu'il n'y a que des auteurs. Dans la chaîne de caractère qu'on souhaite chercher, une étoile « * » peut être saisie pour récupérer tous les coauteurs dans la limite de 100 ou count. Le résultat retourné est un dictionnaire contenant une liste de dictionnaires des auteurs.

La seconde route « /search/publications/<searchString> » permet de rechercher toutes les publications qui contiennent en titre la chaîne de caractère searchString. Au maximum, 100 résultats peuvent être retourné sauf si count indique une limite inférieure à 100. De même que la route précédente, les paramètres « start », « count » et « order » mais aussi la saisie de l'étoile « * » en recherche sont disponibles et fonctionnent de la même manière. Cependant, cette fonction gère aussi un paramètre « filter » qui consiste à filtrer selon d'autres critères comme le nom de l'auteur, le nombre de pages, etc. Ce paramètre est vérifié avant la recherche en vérifiant uniquement la syntaxe, par exemple « filter?author :'James',volume='25' ».

Pour gérer les filtres, il faut d'abord rechercher qu'un titre contenant « searchString » est trouvé. S'il est trouvé, il faut valider tous les filtres en comparant les clés et les valeurs dans le dictionnaire. Si la clé et la valeur correspondent, un compteur s'incrémente. Si ce compteur est égal au nombre de filtre entré, alors la publication est validée. Il faut savoir qu'il n'est pas nécessaire d'avoir la clé et la valeur écrite correctement, il faut uniquement qu'ils soient contenus dans le dictionnaire.

Un problème à faire face lorsqu'il faut ordonner, c'est que toutes les données sont des chaînes de caractères, les nombres inclus. Dans le cas où il n'y a que des nombres, il est possible de convertir la chaîne en entier afin de pouvoir ordonner correctement, sinon il faut ordonner en fonction des caractères. Une vérification est réalisée à la fin de la fonction pour connaître la façon dont il faut ordonner.

Cette seconde route retourne un dictionnaire contenant la liste de dictionnaires sur les publications trouvées.

La route « /authors/<name_origin>/distance/<name_destination> »

Cette route permet de calculer la distance de « name_origin » et de « name_destination » s'il en existe. C'est-à-dire qu'ils doivent être coauteurs pour avoir une distance valable. La fonction gérant cette route vérifie dans un premier temps si les deux noms sont similaires ou non. Si c'est le cas, une distance de 0 et un chemin indiquant « self » est retourné. Sinon, il faut dans un premier temps trouver l'existence des deux auteurs dans une même publication. S'ils sont coauteurs, ils sont ajoutés dans la distance qui les sépare, sachant que par défaut, la distance est de -1 (erreur). Pendant une recherche, si plusieurs distances sont trouvées, la distance la plus courte et le chemin correspondant est retourné.

La route « /error/<code:int>/<msg> »

Cette route est particulière car elle retourne les erreurs rencontrées dans les autres routes, avec un code d'erreur et le message msg indiquant la raison de l'erreur.

API Tester

Un fichier nommé « API_Tester.py » est disponible pour tester globalement et rapidement les fonctionnalités présentes ci-dessus mis à part les erreurs. Le but ici est de faire des requêtes simples pour vérifier les fonctionnalités et de vérifier que toutes les routes fonctionnent.

API Web

Les fonctionnalités disponibles peuvent être testés via url, ce qui n'est pas pratique d'une part parce qu'on ne connait pas toutes les routes, mais d'autre part parce qu'il y a la syntaxe à connaître. C'est ici qu'intervient l'api web, qui offre une interface graphique web pour faire appel aux routes créées. En particulier, on se servira uniquement de quatre routes :

- « /search/publications/<searchString> »
- « /authors/<name>/publications »
- « /authors/<name>/coauthors »
- Et /authors/<name_origin>/distance/<name_destination>

Les options en paramètres sont également disponibles pour les routes qui autorisent les paramètres. Dans le cas où un paramètre est donné à une route ne gérant pas les options, le paramètre est ignoré.

Un bémol présent dans cette interface graphique est le filtre pour la première route. En effet, il faut absolument connaître au minimum la syntaxe particulière «key1 :value1, key2 :value2 »…. Ce qui n'est pas forcément pratique. Ce point aurait pu être amélioré.

Conclusion

Pour conclure, nous avons ici un schéma de type Web client – Web Server – API Server, sans oublier qu'un API Tester est disponible pour tester l'API Server. Le web client permet d'envoyer des requêtes au web server grâce à une interface graphique. Le web server se charge de bien formuler la requête avec les paramètres demandés et fait lui-même une requête à l'API Server qui se chargera de répondre à la requête et d'envoyer la réponse.