

2023 - 2024

RAPPORT SOFT PYTHON



»» Robin MARTIN / Antoine SIRVENT

SOMMAIRE

01 Spécifications

02 Analyse

03 Conception

04 Validation

05 Maintenance

Lien GitHub : <https://github.com/Antoine-SIRVENT/SoftPython>

1 – SPÉCIFICATIONS

Introduction

Le projet vise à développer un moteur de recherche d'information personnalisé, mettant l'accent sur l'autonomie et l'absence de dépendance à des bibliothèques externes. La structuration du code à l'aide de classes et de modules constitue un aspect central pour une gestion optimale du corpus, coeur de notre projet.

Gestion du Corpus

Pour gérer notre corpus, une classe centrale est conçue pour la gestion de celui-ci, offrant des fonctionnalités telles que le chargement de documents, la gestion des mises à jour, et une flexibilité pour l'évolution du corpus au fil du temps.

Moteur de Recherche

Le moteur de recherche, élément central du projet, repose sur la présence de mots-clés dans les documents. Une méthode dédiée est intégrée dans un des fichier pour effectuer des recherches au sein du corpus, permettant à l'utilisateur de formuler des requêtes et d'obtenir des résultats pertinents.

Interface Visuelle

Une interface utilisateur graphique est mise en place pour améliorer l'interaction. Cette interface offre à l'utilisateur une plateforme plus sympa que l'invit de commande pour entrer des requêtes, afficher les résultats, et explorer le corpus de manière agréable.

1 – SPÉCIFICATIONS

Structuration du Code

L'organisation en modules distincts et l'utilisation de classes dans des fichiers séparés sont choisis pour améliorer la lisibilité et la maintenabilité du code. La nomenclature est soigneusement choisie, et des commentaires sont ajoutés de manière appropriée pour garantir une compréhension claire du code.

Développement Propre

Le développement du moteur de recherche se veut propre, en évitant l'utilisation de bibliothèques externes au profit d'une compréhension approfondie des mécanismes internes. Le code est documenté de manière exhaustive, facilitant ainsi la compréhension et la collaboration future.

Évolutivité

L'évolutivité du projet est envisagée, permettant une extension future du moteur de recherche pour gérer plusieurs corpus ou évoluer en fonction des besoins changeants au fil du temps. Ces spécifications définissent un cadre solide pour le développement du projet, balisant la route vers un moteur de recherche d'information personnalisé robuste et polyvalent.

2 – ANALYSE

Environnement de travail

Nous avons choisi d'utiliser PyCharm pour notre projet car c'est un IDE super puissant. La gestion des environnements dans PyCharm est simple. On peut créer des environnements virtuels (en passant par Anaconda) sans se prendre la tête, ce qui nous aide à garder notre projet bien organisé et à éviter les soucis de dépendances. En plus, Pycharm est bien intégré avec Git, et c'est donc facile de garder le projet bien propre en terme de dépendances

Les données

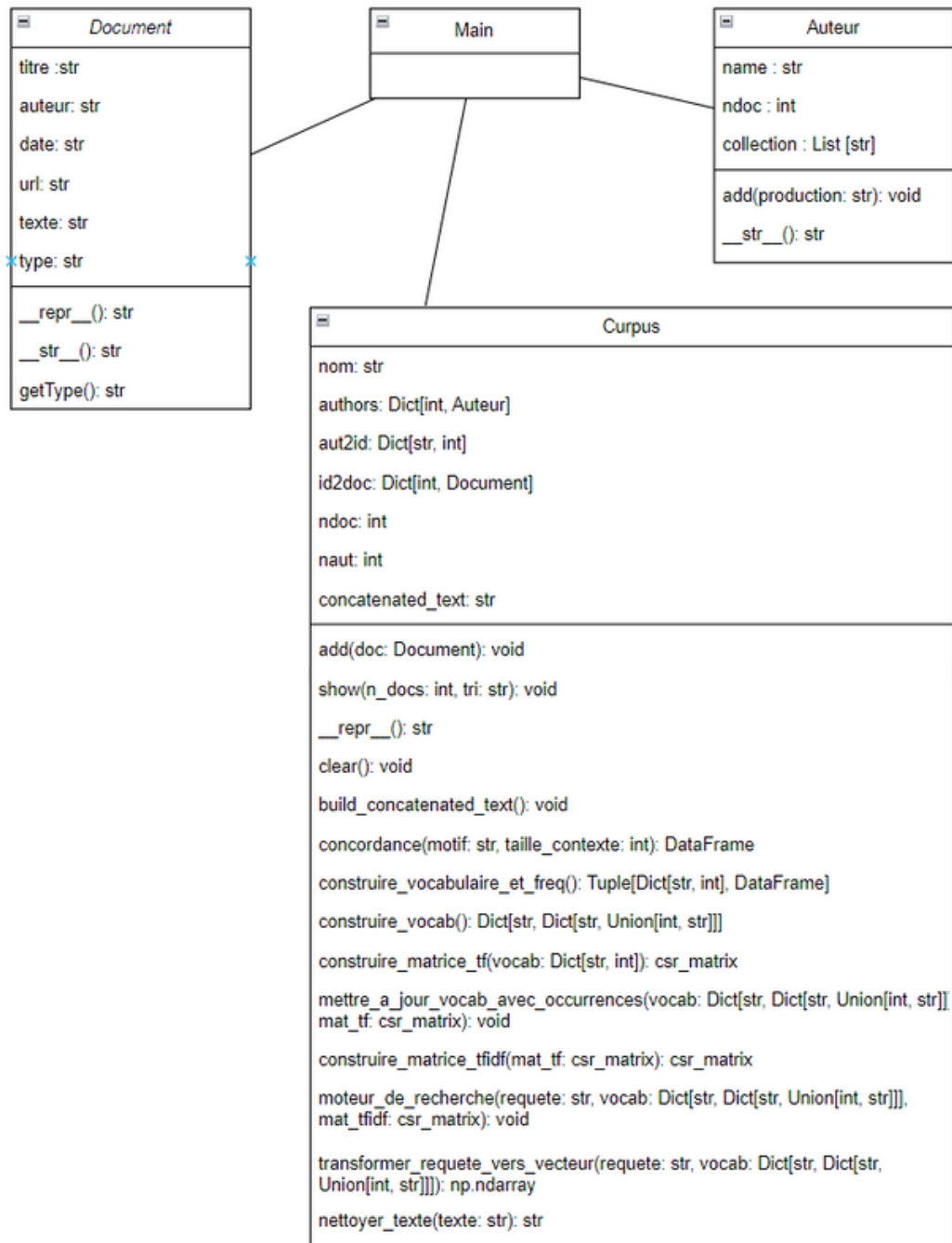
Les données identifiées dans les spécifications incluent principalement une classe Document qui représente des documents génériques. Les attributs de cette classe comprennent *titre*, *auteur*, *datee*, *url*, *texte*, et *type*. Cette classe est la base pour modéliser les différents documents du corpus.

Il y a aussi une classe Auteur utilisée pour représenter les auteurs associés aux documents. Cette classe comprend des attributs tels que le nom de l'auteur, un compteur de documents (ndoc), et une collection de productions (collection).

Ces données nous permettent de structurer et de représenter les documents et les auteurs de manière appropriée, facilitant ainsi notre gestion du corpus et la mise en place du moteur de recherche.

2 - ANALYSE

Diagramme de classes :



3 – CONCEPTION


Partage des tâches

Le partage des tâches a été effectué en fonction de l'avancement des travaux dirigés de chacun ainsi que de nos capacités respectives. Notamment, le code était davantage la responsabilité d'Antoine, tandis que l'interface était celle de Robin.

Problèmes rencontrés

En ce qui concerne la partie affichage, le plus gros problème était le dernier travail dirigé pour trouver l'axe de départ. J'ai rencontré des difficultés avec le TD10 ; je souhaitais effectuer un tri plus précis avec une recherche, mais malheureusement, je n'ai pas réussi à aller jusqu'au bout des choses.

Exemple commenté

C'est très simple, il suffit de lancer le fichier 'TD10.py'. Une fois lancé, après plusieurs secondes, une interface va s'ouvrir. On peut effectuer une recherche dans le corpus et valider la recherche à l'aide du bouton . Ensuite, si l'on veut consulter le corpus, on peut utiliser la liste déroulante pour choisir le tri et cliquer sur le bouton 'Voir le corpus'. Petite note : après le premier lancement, la bibliothèque NLTK 'stopwords' est déjà téléchargée. Ainsi, on peut mettre en commentaire la ligne '6' du document 'corpus.py' pour gagner un peu de temps.

4 - VALIDATION

Tests unitaires :

Nous avons mis en place des tests unitaires pour garantir le bon fonctionnement de différentes parties de notre projet.

Pour la classe Document, nous avons le test `test_get_type` qui vérifie que la méthode `getType` renvoie correctement le type du document. Dans la classe dérivée `ArxivDocument`, le test `test_arxiv_add_auteur` assure que la méthode `ajouter_auteur` fonctionne comme prévu en ajoutant correctement un co-auteur à la liste.

La classe `Auteur` est testée avec `test_add_production` pour s'assurer que la méthode `add` ajoute correctement une production à la collection de l'auteur.

Concernant la classe `Corpus`, deux tests importants sont `test_add_document` qui vérifie que la méthode `add` ajoute correctement un document au corpus, et `test_build_concatenated_text` qui assure que la méthode `build_concatenated_text` construit correctement le texte concaténé du corpus.

Enfin, la classe `CorpusSingleton` est testée avec `test_singleton_instance` pour s'assurer que l'instance du corpus est unique, conforme au modèle singleton.

4 - VALIDATION

Tests globaux :

Lors de l'utilisation de l'interface du programme, nous avons effectué plusieurs tests globaux pour évaluer la fonctionnalité générale.

Nous avons commencé par nettoyer le corpus, puis avons ajouté divers types de documents, y compris des documents Reddit et ArXiv. Nous avons vérifié que ces documents ont été correctement ajoutés et que le nombre total de documents dans le corpus était exact.

La construction de la matrice TF-IDF à partir des documents a également été testée pour garantir son bon fonctionnement sans erreurs.

Le moteur de recherche a fait l'objet de tests approfondis, avec des requêtes variées pour s'assurer que les résultats étaient cohérents et pertinents.

Enfin, nous avons testé la fonctionnalité de sauvegarde et de chargement du corpus dans un fichier binaire, vérifiant que le corpus chargé était identique à celui sauvegardé.

5 – MAINTENANCE

Évolutions Possibles et Facilité d'Implémentation :

Extension des fonctionnalités de recherche :

Il est possible d'ajouter des fonctionnalités avancées comme la recherche sémantique ou contextuelle pour rendre le moteur de recherche plus puissant. Même si cela nécessite une compréhension approfondie des algorithmes de recherche, l'intégration de ces améliorations sera facilitée par la modularité du code existant.

Gestion des documents multilingues :

Si l'application est destinée à un public multilingue, la prise en charge de documents multilingues pourrait être considérablement améliorée. La structure modulaire du code offre une base solide pour cette évolution, bien que cela puisse nécessiter des ajustements dans le traitement du langage naturel.

Optimisation des performances :

Alors que le volume de données augmente, l'optimisation des performances devient cruciale. Cette tâche peut être modérément complexe et nécessite des changements dans le traitement et le stockage des données. L'optimisation des requêtes de recherche et le chargement sélectif de données peuvent être des méthodes efficaces.

5 – MAINTENANCE

Intégration de méthodes d'apprentissage automatique :

Il est possible d'envisager l'intégration de méthodes d'apprentissage automatique pour améliorer la pertinence des résultats de recherche. En raison de la nécessité de données annotées et de l'expertise requise en apprentissage automatique, cela serait plus difficile. Cependant, l'intégration de ces méthodes sera facilitée grâce à l'utilisation du code modulaire.

Système de recommandation :

Un système de recommandation personnalisé peut être une fonctionnalité utile. La difficulté de cette évolution peut varier en fonction de la complexité des algorithmes de recommandation utilisés. Pour introduire des recommandations simples, une approche progressive pourrait être utilisée avant d'étendre progressivement la sophistication de l'algorithme.

En somme, l'avenir du logiciel offre de nombreuses opportunités d'amélioration. La modularité, la flexibilité de la conception et la structure du code existant détermineront souvent la facilité d'implémentation. Pour intégrer progressivement ces évolutions tout en s'adaptant aux retours des utilisateurs et aux tendances technologiques, un processus itératif et agile peut être utilisé.