Justin VILLEROT

Édition · Littératures
Langage · Informatique
Arts · Didactique · Discours

Implémentation d'un système intelligent de recherche d'information

Rapport technique

22/01/2024 - 04/05/2024

ELLIAD, Montbéliard

Thibaud HULIN,
Enseignant Chercheur

Justin VILLEROT

3ème année de BUT Informatique

Jerome HILDENBRAND,
Enseignant référant

Informatique
iut Nord Franche-Comté

Nord Franche-Comté
Belfort - Montbéliard
UNIVERSITÉ DE FRANCHE-COMTÉ

Table des Matières

Introduction **5**

1 Prérequis **6**

1.1 Définitions 6

Intelligence artificielle 6

Word2Vec 6

API / Application Programming Interface 6

1.2 Structure du projet 7

1.3 Bibliothèques et frameworks utilisés 8

2 Explications générales **9**

2.1 Installation 9

Configuration du WordPress 9

Configuration de l'API 11

Importer l'ontologie 13

Vérification 14

2.2 IA Search API 15

Structure fichier 15

2.3 WordPress 17

Structure fichier 17

3 Présentation technique **19**

3.1 API 19

Authentification 19

CRUD 20

Recommandations 22

Query 23

Test 23

Gestion des utilisateurs 23

Modification à faire si changement 24

3.2 WordPress 25

Synchronisation des actions sur les logements 25

Recommandations améliorées par l'IA 26

Autres 27

Table des illustrations **28**

Introduction

Les outils de recommandation actuellement disponibles sur internet ne permettent pas de fournir des recommandations prenant en compte l'intégralité des informations disponibles. L'utilisation de filtres peut masquer des informations qui auraient été pertinentes en fonction de leur configuration.

Prenons l'exemple d'un site de fiches pédagogiques où nous recherchons des documents sur la trigonométrie, le calcul de trajectoire et les polynômes. Il est fort probable que le site ne trouve aucune fiche correspondant à nos attentes, alors même qu'il existe de nombreuses fiches traitant de ces sujets.

Un projet antérieur avait été mené pour répondre à ce problème, utilisant un logiciel basé sur un modèle d'IA de traitement de texte « TF-IDF+Word2Vec » pour recommander des fiches pédagogiques. L'objectif de ce projet était de poursuivre ce travail antérieur en adaptant le système de recommandation pour des logements.

Ainsi, le projet se divise en deux parties :

- La réalisation d'une API encapsulant le projet de recommandation et sa transformation pour traiter des logements.
- L'adaptation d'un site WordPress existant pour utiliser ce nouveau système de recommandation.

Pour finir le projet devait être installer sur un serveur.

1 Prérequis

1.1 Définitions

Intelligence artificielle

L'intelligence artificielle (IA) vise à développer des systèmes capables de reproduire des processus cognitifs humains en utilisant des algorithmes et des modèles mathématiques pour apprendre à partir de données, prendre des décisions et résoudre des problèmes. Son objectif est de créer des machines autonomes capables de penser, apprendre et agir comme des êtres intelligents, imitant ainsi diverses capacités humaines telles que la perception, la compréhension du langage naturel et même l'émotion. Dans notre cas, nous utiliserons un modèle Word2Vec.

Word2Vec

Word2Vec est un modèle de traitement du langage naturel qui utilise des réseaux de neurones pour apprendre des représentations vectorielles de mots à partir de grands corpus de texte. En attribuant à chaque mot un vecteur numérique dense, ce modèle capture les relations sémantiques et syntaxiques entre les mots, permettant ainsi de saisir le contexte et la signification des mots dans un langage donné. Ces vecteurs facilitent l'utilisation des mots dans diverses applications de traitement du langage naturel, telles que la recherche d'informations, la traduction automatique et l'analyse de sentiments.

API / Application Programming Interface

Une API, est un ensemble de règles et de protocoles permettant à des logiciels différents de communiquer entre eux de manière standardisée. Elle permet aux développeurs d'accéder aux fonctionnalités d'un programme ou d'un service sans avoir à connaître sa structure interne. En bref, une API facilite l'intégration entre différentes applications et services.

1.2 Structure du projet

Le projet est divisé en deux applications. Une application Python qui contient une API fastAPI et le modèle IA TF-IDF+Word2Vec, ainsi qu'un serveur Apache qui contient un site WordPress et l'application web PhpMyAdmin pour gérer la BDD[1] MariaDB, comme on peut le voir ci-dessous.

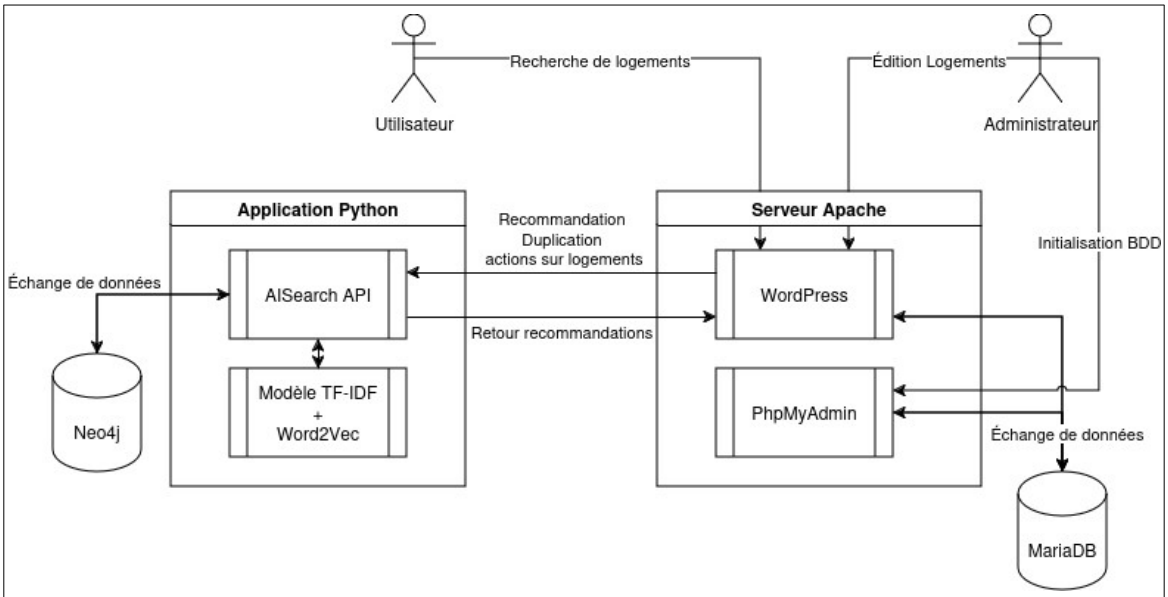

Illustration 1: Schéma structure projet

L'API permet de faire le lien entre l'ontologie Neo4j et le site web WP. Comme nous utilisons WP, nous avons besoin d'une BDD SQL pour stocker les informations nécessaires au fonctionnement du WP.

Notre objectif durant ce projet était d'implémenter l'API et les interactions entre elle et le site web. Nous avons donc décidé de concevoir l'API en premier, car sans l'API impossible de tester les modifications du site web. Nous avons ensuite développé les fonctionnalités du site web qui interagissent avec l'API.

1 BDD – Base de donnée

1.3 Bibliothèques et frameworks utilisés

Pour réaliser ces projets, plusieurs frameworks et plugins associés à ces derniers ont été utilisés.

Frameworks :

- FastAPI est le framework utilisé pour créer l'API.
- WordPress est le framework utilisé pour le site web.

Plugins :

- ACF (Advanced Custom Fields) est un plugin WordPress populaire permettant aux utilisateurs de créer facilement des champs personnalisés pour étendre les fonctionnalités et la flexibilité de leurs sites WordPress sans nécessiter de codage personnalisé.
- CPT UI (Custom Post Type UI) fait référence à un plugin WordPress permettant aux utilisateurs de créer et de gérer des types de publications personnalisés directement depuis l'interface utilisateur de WordPress, éliminant ainsi le besoin de compétences en développement.

2 Explications générales

2.1 Installation

Configuration du WordPress

Apache

Pour installer Apache, exécutez les commandes suivantes :

```
sudo apt update
sudo apt install apache2
```

Après l'installation, vous devez effectuer quelques configurations. Ouvrez le fichier de configuration d'Apache :

```
sudo nano /etc/apache2/apache2.conf
```

Modifiez la ligne "AllowOverride None" en "AllowOverride All", puis redémarrez le serveur Apache avec la commande suivante :

```
sudo systemctl restart apache2.service
```

MariaDB

Pour installer MariaDB, exécutez les commandes suivantes :

```
sudo apt install mariadb-server-10.6 mariadb-server-core-10.6
sudo systemctl start mariadb
sudo systemctl enable mariadb
sudo mysql_secure_installation
sudo mkdir -p /var/lib/mysql/easing
sudo chown -R mysql:mysql /var/lib/mysql/easing
sudo systemctl restart mariadb
```

```
mysql -u root -p
```

Dans l'interface de MariaDB, exécutez les commandes suivantes :

```
CREATE DATABASE easing;
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
GRANT ALL PRIVILEGES ON easing.* TO 'admin'@'localhost';
FLUSH PRIVILEGES;
```

PHP 8.1

Pour installer PHP, exécutez les commandes suivantes :

```
sudo apt update
sudo apt-get install libapache2-mod-php
sudo a2enmod php8.1
sudo systemctl restart apache2.service
sudo apt install phpmyadmin
```

Initialiser la base de données MySQL

Accédez à http://[Votre_adresse_IP_du_serveur]/phpmyadmin/ et importez la base de données de votre ancien site. Ou utiliser le fichier sql fournie dans le projet GitHub.

Dans la table wp_option de la base de données, modifiez les valeurs |home| et |siteurl| pour correspondre à l'URL de votre site Web. Vous devrez peut-être également modifier le format des permaliens dans le panneau d'administration de WordPress.

Panneau d'administration de WordPress :

- http://[Votre_adresse_du_site_WP]/wp-admin

Serveur FTP

Installation du serveur

Pour installer un serveur FTP, suivez les instructions sur le lien suivant ou les commandes ci-dessous :

https://www.it-connect.fr/creation-dun-ftp-avec-utilisateurs-et-repertoire-de-groupe/

```
sudo apt-get install proftpd

sudo nano /etc/proftpd/proftpd.conf
# Décommentez la ligne suivante :
DefaultRoot ~ # cette ligne verrouille l'utilisateur dans son répertoire personnel (home/user), l'empêchant de remonter dans l'arborescence (augmente la sécurité).
sudo systemctl restart proftpd.service
sudo systemctl status proftpd.service
```

Utilisateur pour le serveur FTP

Création de l'utilisateur et configuration de ses droits

```
sudo groupadd WordPress # Créer le groupe WordPress
sudo useradd -s /bin/bash wpuser # Créer l'utilisateur
sudo usermod -aG WordPress wpuser  # Ajouter l'utilisateur au groupe WordPress
sudo passwd wpuser # Définir le mot de passe de l'utilisateur
```

Configuration du répertoire personnel de l'utilisateur dans le serveur Apache

```
cd /var/www/html/ # Accéder à la racine de votre serveur Apache2
sudo mkdir wpuser
sudo chown -R wpuser:WordPress ./wpuser
sudo chmod 775 ./wpuser
sudo usermod -d ./wpuser wpuser # Attribuer le dossier à l'utilisateur
```

Variables d'environnement

Pour que le site WP puisse se connecter à l'API vous devez créer un fichier « .env » dans le dossier « easing » du projet. Vous pouvez trouver la structure du « .env » ci-dessous.

.env

```
LOGIN=<loginAPI>
PASSWD=<PasswordAPI>
API_URL=<addresseAPI>
```

Configuration de neo4j

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
sudo apt-get update
sudo apt-get install neo4j=1:5.11.0
sudo apt-mark hold neo4j # to prevent from changing the neo4j version
sudo apt install openjdk-17-jre
sudo update-java-alternatives --jre --set java-1.17.0-openjdk-amd64
java -version # to check if java version is 17
```

Dans le fichier /etc/neo4j/neo4j.conf des-commentés la ligne suivante :

server.default_listen_address=0.0.0.0

Installer les plugins :

```
mkdir PLDL
cd PLDL
wget "https://github.com/neo4j-labs/neosemantics/releases/download/5.15.0/neosemantics-5.15.0.jar"
wget "https://github.com/neo4j/graph-data-science/releases/download/2.5.7/neo4j-graph-data-science-2.5.7.jar"
sudo mv ./* /var/lib/neo4j/plugins/
sudo neo4j restart
```

Aller sur l'interface web de neo4j http://[your server addres]:7474/browser/

Et connecter vous en utilisant les identifiant de votre serveur, par défault :

- Adress : http://[your server address]:7687
- Username : neo4j
- Password : neo4j

Il vous sera alors demandé de modifier le mot de passe par défaut

Utilisez ensuite ces commandes pour configurer la base de données et la visualiser:

```
CALL n10s.graphconfig.init()
```

Configuration de l'API

```
mkdir IASearchAPI
cd IASearchAPI

# Récupérer le code de l'API
git clone https://github.com/VillerotJustinOrg/IASearchAPI.git

# Configuration de l'environnement virtuel Python
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
nano .env # Configuration du fichier .env voir ci-dessous
```

Importer les Stopwords avec python3

```
import nltk
nltk.download("stopwords")
exit()
```

```
# Déplacer les stopwords nltk au bon endroit
mv ~/nltk .venv/
```

Téléchargement du Modèle

```
mkdir Models # si le répertoire n'existe pas
cd Models
# Téléchargement du modèle, cela peut prendre du temps
wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.fr.300.vec.gz
gunzip cc.fr.300.vec.gz # Décompression du modèle
```

Configuration du service systemctl pour que l'API se lance au démarrage du serveur

```
# Configuration d'un service pour que l'API démarre avec le serveur
nano SearchAPI.sh
nano API.service
# Vous pouvez voir le contenu de ces fichiers ci-dessous
chmod 775 SearchAPI.sh
mv SearchAPI.sh Launchers/
sudo mv API.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl start API.service
sudo systemctl enable API.service
sudo systemctl status API.service
```

Le contenu des fichiers mentionné précédemment se trouve ci-dessous :

Fichier .env

```
APP_NAME="IA Search API"
APP_DESC="API construite pour la recherche améliorée par IA sur Neo4j avec FastAPI"
APP_VERSION="0.1"
DOCS_URL="/docs"
REDOC_URL="/redoc"
NEO4J_URI="bolt://localhost:7687" # URL de votre serveur Neo4j
NEO4J_USERNAME="neo4j"
NEO4J_PASSWORD="password"
DB_PREFIX="" # s'il y a un préfixe avant les labels des nœuds
APP_PASSWORD="secure_this" # Mot de passe de l'API pour créer le premier utilisateur
SECRET_KEY="secret_key"
ALGORITHM="HS256"
ACCESS_TOKEN_EXPIRE_MINUTES=10080 # Durée de vie du jeton d'authentification
LOAD_LOCAL_MODEL=true
```

Le fichier .env doit être placé dans le dossier easing du projet GitHub de l'API

Vous pouvez créer un premier utilisateur avec la requête suivante POST /auth/launch_user plus d'information ICI.

Fichier SearchAPI.sh

```
#!/bin/bash
# Changer de répertoire vers le dossier spécifié
cd /home/jvillero/IASearchAPI
# Exécuter la commande
.venv/bin/python app/main.py
```

Fichier API.service

```
[Unit]
Description=My API
After=multi-user.target
Conflicts=getty@tty1.service

[Service]
Type=simple
ExecStart=[Chemin vers le fishier SearchAPI.sh]
StandardOutput=file:/home/user/logs/API.log
StandardError=file:/home/user/logs/API_Error.log
StandardInput=tty-force

[Install]
WantedBy=multi-user.target
```

Importer l’ontologie

Importer une dump de l’ontologie

Pour importer un dump de l'ontologie, utilisez la commande suivante :

```
sudo neo4j-admin database load --overwrite-destination --from-path=./ neo4j
```

Si vous importez une ontologie notée que les utilisateurs de l’API correspondront à ceux dans l’ontologie que vous avez importé.

Exporter le dump de l’ontologie

Si vous avez déjà une base de données neo4j fonctionnelle et que vous souhaitez la transférer, utilisez les commandes suivantes pour exporter un dump de la base de données :

```
sudo neo4j-admin database dump neo4j --to-path=/home/user/
```

Attention : Si vous utilisez Neo4j Desktop, vous devez utiliser le noe4j-admin qui s'y trouve, et non celui qui est installé sur l'ensemble du système. Additionnellement noter que neo4 dois être arrêter pour pouvoir effectuer des actions sur l’ontologie.

Vérification

Nous pouvons utiliser la commande suivante pour vérifier si toutes les applications sont en cours d'exécution et écoutent sur le bon port :

```
sudo netstat -tulpn
```

Nous devrions avoir un résultat similaire à ceci :

```
Proto   Recv-Q  Send-Q  Local Address       Foreign Address     State       PID/Program name
tcp     0       0       127.0.0.53:53       0.0.0.0:*           LISTEN      1079/systemd-resolv
tcp     0       0       127.0.0.1:3306      0.0.0.0:*           LISTEN      1239/mariadbd
tcp     0       0       0.0.0.0:8000        0.0.0.0:*           LISTEN      1918/.venv/bin/pyth
tcp     0       0       0.0.0.0:22          0.0.0.0:*           LISTEN      6867/sshd: /usr/sbi
tcp     0       0       0.0.0.0:631         0.0.0.0:*           LISTEN      1958/cupsd
tcp6    0       0       :::7474             :::*                LISTEN      14696/java
tcp6    0       0       :::7687             :::*                LISTEN      14696/java
tcp6    0       0       :::21               :::*                LISTEN      12225/proftpd
tcp6    0       0       :::22               :::*                LISTEN      6867/sshd: /usr/sbi
tcp6    0       0       :::80               :::*                LISTEN      6884/apache2
tcp6    0       0       :::631              :::*                LISTEN      1958/cupsd
...
```

Illustration 2: Résultat commande netstat

Nous devons vérifier si les programmes suivants sont présents :

- 127.0.0.1:3306 0.0.0.0:* LISTEN [PID]/MariaDB | Base de données MariaDB
- 0.0.0.0:8000 0.0.0.0:* LISTEN [PID]/.venv/bin/pyth | API Python
- :::7474 :::* LISTEN [PID]/java | Navigateur Web Neo4j
- :::7687 :::* LISTEN [PID]/java | Base de données Neo4j
- :::21 :::* LISTEN [PID]/proftpd | Serveur FTP
- :::80 :::* LISTEN [PID]/apache2 | Serveur Apache2

Nous pouvons aussi vérifier si les documentations de l'API sont présentes en nous rendant sur les adresses suivantes :

- http://[Votre_adresse_du_serveur]:8000/docs
- http://[Votre_adresse_du_serveur]:8000/redoc

2.2 IA Search API

L'API conçue pour la recherche améliorée par l'intelligence artificielle sur Neo4j avec FastAPI est une interface programmable qui intègre les fonctionnalités avancées de Neo4j, une base de données orientée graphe, avec les performances et la flexibilité de FastAPI, un framework web rapide pour Python.

Cette API permet aux chercheurs d'exploiter les capacités puissantes de Neo4j pour l'analyse de données complexes, tout en profitant des fonctionnalités de développement rapide et de haute performance de FastAPI.

Grâce à cette combinaison, les utilisateurs peuvent créer des applications de recherche sophistiquées, exploitant les avantages de la modélisation de données en graphe pour obtenir une meilleure compréhension de leurs ensembles de données.

Structure fichier

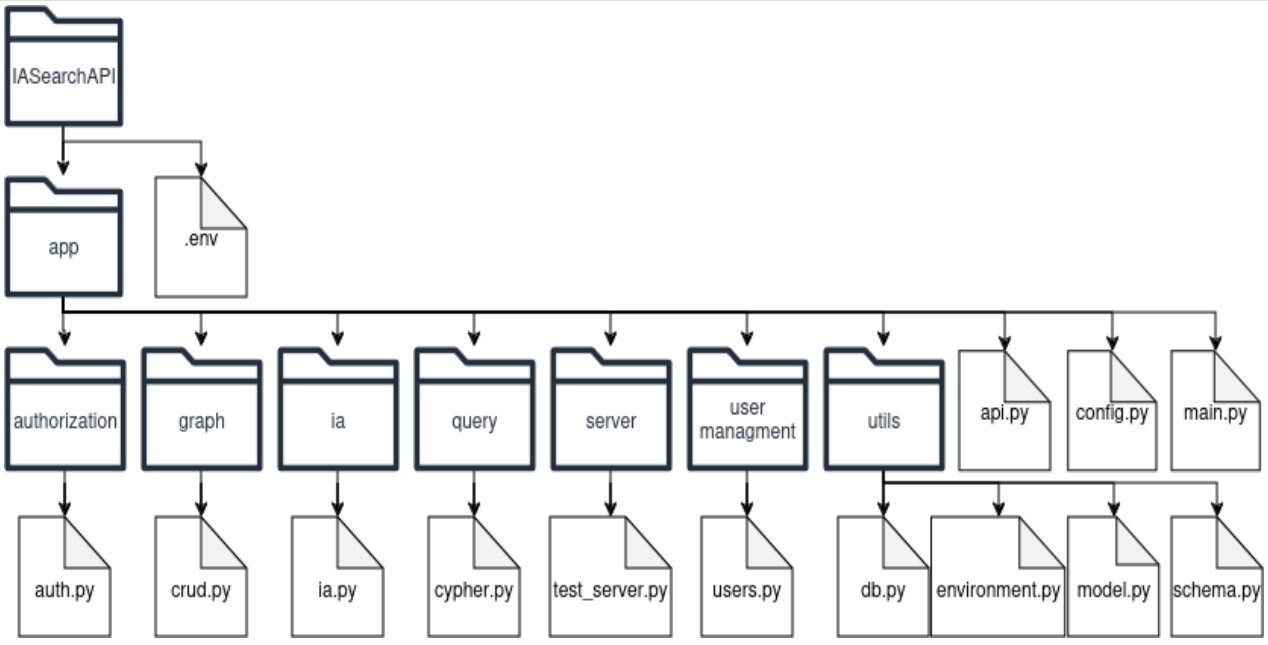

Illustration 3: Arborescence des fichiers API

Dans la structure ci-dessus, au niveau de la racine du dossier de l'API, vous remarquerez un répertoire contenant le code de l'API ainsi qu'un fichier .env qui renferme différentes variables d'environnement. Il est à noter que le fichier .env n'est pas présent par défaut et doit être ajouté manuellement.

Dans le répertoire "app", nous trouverons trois fichiers Python : "api.py", qui crée l'application FastAPI, "config.py", contenant des informations pour le démarrage de l'API, et "main.py", contenant le script de lancement de l'application. De plus, plusieurs dossiers sont présents, chacun correspondant à une route de l'API, à l'exception du dossier "utils", qui regroupe des bibliothèques de fonctions. Dans ce dossier "utils", nous pourrons découvrir quatre fichiers Python : "db.py" pour établir la connexion avec l'ontologie Neo4j, "environnement.py" pour accéder au contenu du fichier ".env", "model.py" pour charger le modèle Word2Vec, et enfin "schema.py" pour le schéma de données d'un nœud.

Voici à quelle route quel dossier correspond :

- "autorisation" → /auth
- "graph" → /graph
- "ia" → /ia
- "query" → /q
- "server" → /test
- "user_managment" → /users

2.3 WordPress

Étant donné que nous utilisons le plugin ACF pour faire les formulaires, nous avons donc dû utiliser les « hooks » de WP pour récupérer les modifications faites sur les logements dans le WP et transposer ces modifications dans la base de données Neo4j.

Le code du site peut-être trouvé dans le dossier easing du projet GitHub **Easing-wp**.

Dans le dossier du site seul le dossier wp-content/theme/easting/ contient du code lié au projet les autres dossiers contiennent uniquement des fichiers de fonctionnement WP.

Structure fichier

Comme nous pouvons le voir ci-contre la majorité des fichiers du site web sont des fichiers gérés par WordPress (fichier en rouge dans le schéma). Seul le fichier .env à la racine du projet et le contenu du dossier easing sont pertinents en ce qui concerne le code du site web.

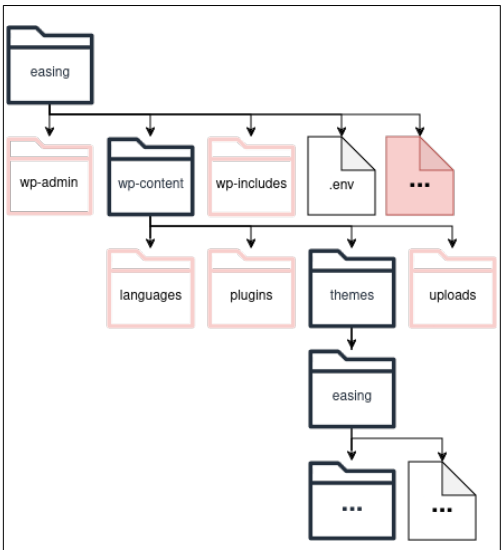

Illustration 4: Arborescence WP

Notez que le fichier .env n'est pas présent dans le GitHub du projet et qu'il faudra donc le rajouter manuellement.

Étant donné que la majorité des fichiers liés au site se trouvent dans le dossier du thème "easing", nous allons expliquer la structure de fichiers de ce thème.

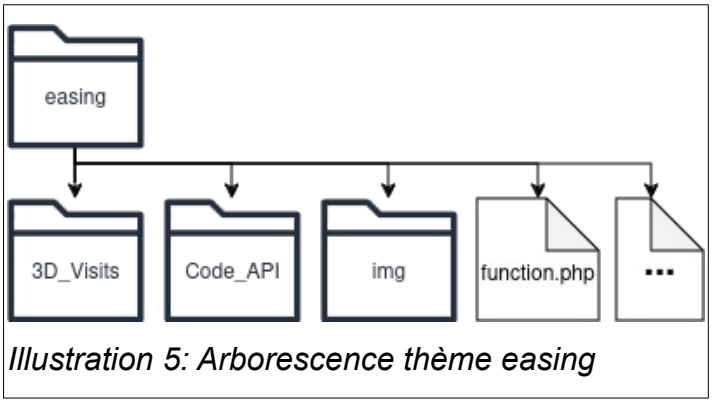

Illustration 5: Arborescence thème easing

Le thème "easing" comporte trois dossiers :

1. "3D_Visits" : contient les mini-sites web obtenus lors de la décompression des archives de visite virtuelle fournies pendant l'édition d'un logement.
2. "Code_API" : contient le code responsable de la synchronisation des actions sur les logements.
3. "img" : contient les images utilisées dans le thème, notamment les différents icônes et logos.

Le thème contient également plusieurs fichiers, mais nous ne parlerons que du fichier "functions.php", qui utilise une fonctionnalité de WordPress permettant d'intercepter l'exécution de fonctions WP. Ce fichier ne doit donc pas être renommé. Plus de détails sur les autres fichiers seront donnés dans l'explication technique du site web.

3 Présentation technique

3.1 API

L'API peut être divisée en six routes principales :

1. Authentification pour se connecter à l'API.
2. CRUD Neo4j pour gérer les nœuds et les relations dans la BDD Neo4j.
3. Recommandation IA, recommandation de logement par IA.
4. Query pour effectuer des requêtes plus complexes dans Neo4j.
5. Test pour vérifier le bon fonctionnement de l'API.
6. Gestion des utilisateurs pour gérer qui a accès à l'API.

Authentification

POST /auth/token

Permet d'obtenir un token d'authentification qui peut être utilisé pour accéder à la plupart des routes disponibles dans l'API. Pour ce faire, vous devez placer le token reçu dans l'en-tête de votre requête. Il est à noter que cette requête, contrairement aux autres, utilise x-www-form-urlencoded et non JSON pour l'encodage du corps de la requête.

Le corps de la requête doit contenir les informations suivantes :

- "username" : l'identifiant de l'utilisateur API auquel vous voulez vous connecter.
- "password" : le mot de passe du même utilisateur.

POST /auth/launch_user

?username=<S>&password=<S>&application_password=<S>

Permet de créer un utilisateur de l'API en utilisant le code secret de l'API.

CRUD

Neo4j offre une flexibilité exceptionnelle dans la structuration des données, permettant la création de sous-types de nœuds et facilitant ainsi l'adaptabilité des schémas de données.

Cette base de données orientée graphe repose sur des nœuds, des arêtes et des propriétés pour modéliser efficacement des relations complexes, simplifiant la manipulation des données grâce à Cypher[2].

Avec des performances remarquables et une capacité d'évolutivité, Neo4j excelle dans l'analyse de réseaux et la découverte de motifs complexes, offrant une interface conviviale pour la création et la gestion des bases de données en graphe.

C'est pourquoi un simple CRUD pour les nœuds et un second pour les relations suffisent à exploiter pleinement ses fonctionnalités.

Notez que toutes les sous-routes de la route /graph requièrent un header contenant le token obtenu en s'authentifiant.

Nœuds

POST /graph/create_node?label=<string>

Permet de créer un nœud avec un label donné, un JSON peut être fourni dans le body de la requête pour définir les propriétés du nœud.

GET /graph/read/{node_id}

Permet d'obtenir les informations d'un nœud d'identifiant donné.

GET /graph/read_node_collection?search_node_property=<string>&node_property_value=<string>

Permet d'obtenir les informations de nœuds avec une propriété avec une valeur donnée.

PUT /graph/update/{node_id}

Permet de mettre à jour un nœud avec un identifiant donné. Un JSON peut être fourni dans le corps de la requête pour définir les propriétés du nœud.

2 Cypher – Langage de Query propre à Neo4j

POST /graph/delete/{node_id}

Permet de supprimer un nœud d'identifiant donné.

Relations

POST /graph/create_relationship?label=<string>

Permet de créer une relation avec un label donnée, un JSON peut-être fourni dans le body de la requête pour définir les propriétés du nœud, ainsi que les nœuds qui composent la relation. Le JSON doit être formaté de la façon suivante :

```
{
        "relationship_type": "Owned_by",
        "relationship_attributes":{},
        "source_node_id": 10,
        "target_node_id": 13
}
```

Illustration 6: JSON exemple création relation

GET /graph/read_relationship/{relationship_id}

Permet d'obtenir les informations d'une relation d'un ID donnée.

PUT /graph/update_relationship/{relationship_id}

Permet de mettre à jour une relation avec un identifiant donné, un JSON peut-être fourni dans le corps de la requête pour définir les propriétés de la relation.

POST /graph/delete_relationship/{relationship_id}

Permet de supprimer une relation d'identifiant donné.

Utilité

Certaines requêtes qui sont souvent utilisées ou des variantes de certaines requêtes ont été ajoutées pour faciliter l'utilisation de l'API, et limiter l'utilisation de la requête Query.

GET /graph/read_relationship_node_label/?node_id=<int> ?_label=<string>

Permet d'obtenir toutes les relations d'un certain type qui sont reliées à un nœud donné.

GET /graph/read_relationship_btwn_node/?node_id1=<int> ?node_id2=<int>

Permet d'obtenir toutes les relations entre deux nœuds donnés.

POST /graph/delete_all_relationship/{node_id}

Permet de supprimer toutes les relations d'un nœud d'identifiant donné.

Recommandations

POST /IA/search

Interroger la base de données pour récupérer une liste de logement en fonction des filtres donnés dans le corps de la requête, puis faire une recommandation parmi les logements valide grâce à l'IA.

Cette fonction est divisée en quatre sous-fonctions, premièrement la fonction AISearch_part1 permet de traiter les informations fournies dans le corps de la requête. Pour fonctionner la requête a besoin d'un JSON contenant les informations suivantes :

- destination=<string>
- start=<YYYY-MM-DD>
- end=<YYYY-MM-DD>
- number_person=<int>
- adult=<int>
- kid=<int>
- baby=<int>
- pet=<int>
- n_result=<int>
- method=<string> "TF-IDF+Word2Vec" ou "TF-IDF"
- similarity_method=<string> "cosine" ou "euclidean" ou "dot"
- user_request=<string>

Certains filtres supplémentaires peuvent être ajoutés dans l'objet advanced_filters :

- Min_Price=<int>
- Miax_Price=<int>
- type=<string>
- categorie=<string>
- reservation=<string> "on" if true
- arrive_auto=<string> "on" if true
- nbr_chambre=<int>
- nbr_salle_bain=<int>
- nbr_lits=<int>

Ensuite, l'API utilisera les divers filtres fournis pour extraire une liste de logements à l'aide d'une requête Cypher.

Une fois cette liste de logements obtenue, l'API concaténera toutes les informations pertinentes relatives à chaque logement dans un texte unique par logement.

Ensuite, l'API fournira cette liste de textes au modèle TF-IDF+Word2Vec.

Le modèle comparera alors ces textes avec le texte de la requête utilisateur et les classera en mettant en avant ceux qui correspondent le mieux.

Enfin, l'API traitera les résultats renvoyés par le modèle et les fera correspondre avec les logements de la base de données.

Query

POST /q

Interroger la base de données à l'aide d'une chaîne de caractères Cypher personnalisée dans le corps de la requête.

Test

GET /test/info

Retourne des informations à propos de l'API

GET /test/neo4j

Retourne des informations concernant la connexion de l'API à la BDD neo4j

GET /test/up

Retourne le message « the server is up » si l'API et bien démarrer.

Gestion des utilisateurs

GET /users/me

Retourne les informations de l'utilisateur associé au token utilisé pour se connecter.

GET /users/{username}

Retourne les informations de l'utilisateur associé au nom d'utilisateur donné.

POST /users/create?username=admin&password=admin

Créer un nouvel utilisateur en fonction des informations fournies dans le corps de la requête.

PUT /users/{username}/update

Met à jour un utilisateur en fonction des informations fournies dans le corps de la requête. Le corps de la requête doit suivre le format suivant :

```
{
        "username":"superadmin",
        "full_name":"Changed Administrateur"
}
```

Illustration 7: JSON exemple update user

DELETE /users/{username}/delete

Supprime l'utilisateur de nom d'utilisateur donné.

PUT /users/me/reset_password

Permet de changer le mot de passe de l'utilisateur associé au token utilisé pour cette requête.

Modification à faire si changement

Pour mettre à jour l'API et faire en sorte qu'elle puisse fonctionner avec une nouvelle structure de données ou des nouveaux filtres, nous devons modifier trois fonctions.

Premièrement la fonction **AISearch_part1** pour prendre en compte les modifications des paramètres fournis dans le corps de la requête. Ensuite la fonction **AISearch_part2** pour changer la requête Cypher qui récupére la liste de logement. Et enfin la fonction **node_texte_concatenation** pour changer les informations qui vont être ajoutées dans le texte traité par l'IA.

3.2 WordPress

Le code du site web peut être divisé en trois parties. La première partie concerne tout le code lié à la synchronisation des actions sur les logements entre la base de données SQL du site et l'ontologie Neo4j de l'API. La deuxième partie comprend tout le code lié à l'amélioration de la recommandation par l'IA. Enfin, la troisième partie concerne le code lié à l'affichage des logements.

Synchronisation des actions sur les logements

Pour le code lié à la synchronisation des actions sur les logements les fichiers qui vont nous intéresser sont, les fichiers du dossier "Code_API" et le fichier "functions.ph".

Le fichier "functions.ph" est là où tout commence, il utilise les "hooks" de WP qui permettent d'intercepter l'appel d'une fonction et de rediriger les informations envoyées à la fonction d'origine vers une fonction donnée. C'est grâce à cette fonctionnalité que nous pouvons dupliquer les actions effectuées sur les logements et les appliquer sur l'ontologie.

Le "hook" va donc intercepter la fonction de modification d'un post et envoyer les informations à la fonction "send_data_to_api_on_post_save".

La fonction "send_data_to_api_on_post_save" va vérifier si le statut de la modification n'est pas un brouillon et si ce n'est pas le cas, elle va ensuite rediriger les informations obtenues vers la fonction Routeur associée au type de poste. Notez que certains postes ne requièrent pas de traitement spécifique et utilisent donc le Routeur par défaut.

Ensuite dans le dossier "Code_API" se trouve les différents fichiers associés à leur type de post, on y trouve aussi un fichier "UtilsAPI" qui contient une bibliothèque de fonction faisant des requêtes à l'API.

Les fichiers associés à un type de poste se structurent de cette manière :

- Routeur_"type de post" qui va vérifier si le nœud existe dans l'ontologie et en fonction des informations reçues va appeler la fonction adaptée.

- get_"type de post" qui permet d'obtenir le poste associé dans neo4j
- get_"type de post"_id qui permet d'obtenir l'ID du poste associé dans neo4j
- create_"type de post" qui crée un nœud dans l'ontologie
- update_"type de post" qui met à jour le nœud associé
- delete_"type de post" qui supprime le nœud associé
- Delete_All_Relationship qui supprime toutes les relations du nœud associé

Concernant le fichier "UtilsAPI", il comprend les fonctions suivantes :

- "get_API_Token" : permet de récupérer le token de connexion de l'API.
- "add_field_info_to_body" : construit le corps de la requête de création et d'édition de nœud dans l'ontologie Neo4j.
  - "body_builder" : sous-fonction de "add_field_info_to_body".
- "update_relationship" : met à jour une relation entre deux nœuds d'ID et de label donné
  - "update_relationship_between_node" : sous-fonction
  - "check_relationship_between_node" : vérifie si une relation entre deux nœuds donnés existe.
  - "create_relationship_between_node" : créer une relation entre deux nœuds
  - "delete_old_relationships" : supprime toutes les relations d'un nœud qui n'est pas dans une liste donnée
- "delete_relationship" : supprime une relation d'ID donné.

Recommandations améliorées par l'IA

Les fichiers liés aux recommandations améliorées par l'IA peuvent-être séparées en deux parties, les fichiers liés au formulaire, et ceux liés au traitement des résultats du formulaire. Les fichiers "adaptation.json", "equipment.json", "Advanced_Filters.css", "tabs.js", et "FilterForm.php" appartiennent à la première partie et "Filter-Treatment.php" appartient à la seconde.

En ce qui concerne le formulaire, le fichier "FilterForm.php" contient le code html et php du formulaire. La partie filtre avancé nécessite cependant les fichiers suivants : "Advanced_Filters.css" qui contient le CSS des onglets des filtres avancées, et "tabs.js" qui contient le code javascript qui fait fonctionner les onglets et

qui permet de transformant les fichiers json "adaptation.json" et "equipment.json" en liste de case à cocher.

Pour le traitement du filtre tout se passe dans le fichier "Filter-Treatment.php". Tout d'abord les différents éléments du filtre sont récupérés de la variable $_POST de php et elles sont stockées dans des variables.

C'est à ce moment que deux différents traitements peuvent-être appliqués si parmi les filtres il y a une requête utilisateur, ou une date de début ou de fin. Les informations du formulaire seront traitées par l'API, sinon elles seront traitées en utilisant les filtres de WP.

En ce qui concerne le traitement par l'API le code va construire une requête pour l'API et une fois la réponse obtenu, elle va récupérer les identifiants du post associer aux différents logements reçus, et récupérera les logements associés, stockés dans la BDD du WP.

Autres

Les autres fichiers dans le dossier thème comprennent les différents fichiers qui composent les pages html du site web, un fichier CSS pour les styles du site, et un fichier main.js qui permet de faire l'affichage des différentes cartes de localisation des logements ainsi que quelques fonctions javascript comme le traitement de l'ajout de nouvelles locations.

Les fichiers header.php et footer.php contiennent respectivement l'entête et le pied de page :

- Le fichier index.php la page d'accueil
- template-logements.php la page de résultat du filtre
- Le fichier single-logement.php la page individuelle des logements.

Table des illustrations

Illustration 1: Schéma structure projet.....7
Illustration 2: Résultat commande netstat.....14
Illustration 3: Arborescence des fichiers API.....15
Illustration 4: Arborescence WP.....17
Illustration 5: Arborescence thème easing.....17
Illustration 6: JSON exemple création relation.....21
Illustration 7: JSON exemple update user.....24

Table des matières

Introduction **5**

1 Prérequis **6**

1.1 Définitions 6

Intelligence artificielle 6

Word2Vec 6

API / Application Programming Interface 6

1.2 Structure du projet 7

1.3 Bibliothèques et frameworks utilisés 8

2 Explications générales **9**

2.1 Installation 9

Configuration du WordPress 9

Apache 9

MariaDB 9

PHP 8.1 10

Initialiser la base de données MySQL 10

Serveur FTP 10

Installation du serveur 10

Utilisateur pour le serveur FTP 10

Variables d'environnement 11

.env 11

Configuration de l'API 11

Fichier .env 12

Fichier SearchAPI.sh 13

Fichier API.service 13

Importer l'ontologie 13

Exporter une dump de l'ontologie 13

Exporter le dump de l'ontologie 13

Vérification 14

2.2 IA Search API 15

Structure fichier....15
2.3 WordPress....17
Structure fichier....17
3 Présentation technique....19
3.1 API....19
Authentification....19
CRUD....20
Nœuds....20
POST /graph/create_node?label=<string>....20
GET /graph/read/{node_id}....20
GET /graph/read_node_collection?search_node_property=<string>&node_property_value=<string>....20
PUT /graph/update/{node_id}....20
POST /graph/delete/{node_id}....21
Relations....21
POST /graph/create_relationship?label=<string>....21
GET /graph/read_relationship/{relationship_id}....21
PUT /graph/update_relationship/{relationship_id}....21
POST /graph/delete_relationship/{relationship_id}....21
Utilité....21
GET /graph/read_relationship_node_label/?node_id=<int> ?_label=<string>....21
GET /graph/read_relationship_btwn_node/?node_id1=<int> ?node_id2=<int>....22
POST /graph/delete_all_relationship/{node_id}....22
Recommandations....22
POST /IA/search....22
Query....23
POST /q....23
Test....23
GET /test/info....23
GET /test/neo4j....23
GET /test/up....23

Gestion des utilisateurs....23
GET /users/me....23
GET /users/{username}....24
POST /users/create?username=admin&password=admin....24
PUT /users/{username}/update....24
DELETE /users/{username}/delete....24
PUT /users/me/reset_password....24
Modification à faire si changement....24
3.2 WordPress....25
Synchronisation des actions sur les logements....25
Recommandations améliorées par l'IA....26
Autres....27
Table des illustrations....28