Année scolaire 2024-2025
dans le cadre du projet SAE du Semestre 5

Dossier de programmation

Création du site de recherche et de recommandation de séries

Ce rapport de programmation a été réalisé par Cédric Longuet et par Valentane Vacquié, non-alternant.

iut TOULOUSE
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

Sommaire

Introduction

L'objectif principal de ce projet est de proposer un service web qui a deux fonctions. Premièrement, ce service devra répondre à la fonction de recherche de séries en prenant en compte une liste de mots-clefs prédéterminés par l'utilisateur. Deuxièmement, le service web devra proposer un système de recommandation qui prendra en compte les évaluations octroyées aux séries par l'utilisateur. Dans le cadre de ce projet, plusieurs outils ont été utilisés.

Ce projet sera remis sur une machine virtuelle pour garantir une reproductibilité et une portabilité optimales. Afin de gérer les différents composants de l'application, Docker Compose sera utilisé. Cet outil permet de définir et de gérer plusieurs conteneurs Docker, chacun exécutant une partie spécifique de l'application, tout en simplifiant les processus de configuration et de déploiement. Grâce à Docker Compose, l'ensemble de l'application peut être déployé de manière cohérente et reproductible sur différentes machines. De plus, l'utilisation de Docker Compose permet une uniformité des environnements de développement.

Avant d'intégrer les données dans l'application, une étape cruciale de prétraitement a été réalisée. Cette phase, entièrement développée en Python, a permis de nettoyer et structurer les fichiers de sous-titres des séries TV. Ce prétraitement garantit des résultats de recherche plus précis et un fonctionnement optimal des outils d'analyse.

Le cœur de l'application repose sur un script Python, utilisant le framework Flask. Flask est un micro-framework léger et flexible qui facilite la création d'applications web en Python. Il permet de structurer le code de manière modulaire tout en offrant une grande liberté pour personnaliser le comportement du serveur. Flask a également été utilisé pour gérer les sessions utilisateurs, offrant ainsi une gestion sécurisée des informations de session, comme les préférences des utilisateurs.

L'interface utilisateur de l'application a été réalisée avec une combinaison de HTML, CSS, et JavaScript. Le JavaScript, écrit avec jQuery, permet de dynamiser l'interface et d'ajouter des fonctionnalités interactives côté client. Grâce à cette technologie, des mises à jour de contenu peuvent être effectuées sans recharger la page, offrant ainsi une expérience utilisateur plus fluide et réactive.

Dans ce dossier, nous allons explorer les différentes étapes de la conception de cette application et les choix techniques effectués.

I - Architecture générale du dossier SAE_S5

flask	api	.env
		api.py
		tf_idf.py
	static	allSeries.js
		index.js
		recommandation.js
		styles.css
	templates	allSeries.html
		index.html
		recommandation.html
	manage.py	
python_nettoyage	0_main.py	
	a_unzip.py	
	b_clean.py	
	c_combine.py	
	d_clean_VO_VF.py	
	del_vofSRT.py	
	libRecode.py	
	libSrt.py	
sous-titres	ex : 24	vf.txt
		vo.txt
	ex : bones	vf.txt
		vo.txt
.gitignore		
docker-compose.yml		
Dockerfile		
README.md		

requirements.txt
tfidf_matrix.pkl
vectorizer.pkl

II - Nettoyage des données brutes

Cette étape est fondamentale pour établir une base de données propre et exploitable. Elle est essentielle au bon fonctionnement des fonctionnalités de recherche et de recommandation, qui sont au cœur de l'efficacité du site web.

Le nettoyage des données se fait dans le sous-dossier nommé "python_nettoyage". Comme son nom l'indique, ce dossier contient plusieurs fichiers python dédiés à ce processus.
Chaque fichier comprend plusieurs petites fonctions et une fonction principale qui combine leurs utilisations pour réaliser une étape du nettoyage. Le fichier 0_main.py fait appel a chacune de ces fonctions principales pour exécuter l'ensemble du nettoyage.

Analysons de plus près les fonctions principales de chacun de ces fichiers.

A) Fichier a_unzip.py

Ce fichier est un peu particulier car il comprend deux fonctions principales.

1. Fonction unzip(main_directory)

```python
def unzip(main_directory):
    for dossier in os.listdir(main_directory):
        full_dossier_path = os.path.join(main_directory, dossier)
        if os.path.isdir(full_dossier_path):
            unzip_files(full_dossier_path)
            move_files(full_dossier_path)
            delete_unwanted_files(full_dossier_path)
            changeToSrt(full_dossier_path)
            change_encoding(full_dossier_path)
            delete_empty_folders(full_dossier_path)
```

↳ Dézipper les fichiers zippés
↳ Supprimer les fichiers vides
↳ Transformer les fichiers textes (txt) en fichiers sous-titres (srt)
↳ Changer l'encodage en latin-1
↳ Organisez chaque série dans un unique dossier contenant uniquement des fichiers, sans inclure de sous-dossiers

2. Fonction clean_folder_name(directory)

```python
def clean_folder_name(directory):
    for root, dirs, files in os.walk(directory, topdown=False):
        for folder in dirs:
            new_folder = wordninja.split(folder)
            new_folder = '_'.join(new_folder)
            try:
                os.rename(os.path.join(root, folder), os.path.join(root, new_folder))
                print(f"Renommage du dossier {folder} en {new_folder}")
            except Exception as e:
                print(f"Erreur lors du renommage du dossier {folder} en {new_folder}: {e}")
    clean_folder_name_exception(directory)
```

↳ Normaliser les noms des dossiers (noms des séries)

B) Fichier b_clean.py

1. Fonction clean(st_dir, series)

```python
def clean(st_dir, series):
    for serie in series:
        print(f"Nettoyage de la série {serie}")
        if not os.listdir(os.path.join(st_dir, serie)):
            os.rmdir(os.path.join(st_dir, serie))
            continue
        for episode in lib.get_srt(st_dir, serie):
            supprimer_lignes_vides(st_dir, serie, episode)
            supprimer_retour_ligne(st_dir, serie, episode)
            supprimer_balisage(st_dir, serie, episode)
            print(f"Nettoyage de {episode} terminé")
```

↳ Supprimer les lignes vides dans les fichiers sous-titres
↳ Supprimer les retours à la ligne dans les fichiers sous-titres
↳ Supprimer les balises dans les fichiers sous-titres
↳ Supprimer les dossiers vides

C) Fichier c_combine.py

1. Fonction combine(st_dir, series)

```
def combine(st_dir, series):
    combine_VO(st_dir, series)
    combine_VF(st_dir, series)
    combineChuck(st_dir)
```

↳ Rassembler tous les sous-titres identifiés comme version originale dans un fichier texte unique nommé “vo.txt”, et ce, pour chaque série
↳ Rassembler tous les sous-titres identifiés comme version française dans un fichier texte unique nommé “vf.txt”, et ce, pour chaque série
↳ Supprimer les anciens fichiers de sous-titres qui ne s’appellent ni vo.txt ni vf.txt

D) Fichier d_clean_VO_VF.py

1. Fonction clean_VO_VF(st_dir, series)

```
def clean_VO_VF(st_dir, series):
    for serie in series:
        for episode in os.listdir(f'{st_dir}/{serie}'):
            if episode.endswith('.txt'):
                with open(f'{st_dir}/{serie}/{episode}', 'r', encoding='latin-1') as file:
                    contenu = file.read()
                contenu = clean(contenu)
                with open(f'{st_dir}/{serie}/{episode}', 'w', encoding='latin-1') as file:
                    file.write(contenu)
                    print(f"Nettoyage du fichier {episode} de la série {serie}")
```

↳ Parcourir les fichiers textes de chaque série
↳ Lire le contenu de ces fichiers
↳ Nettoyer le texte en supprimant des éléments inutiles tels que les pipes (|), les accolades, les parenthèses, les espaces superflus, ainsi que d'autres caractères indésirables
↳ Réécrire le texte nettoyé dans les mêmes fichiers

E) Fichier del_vofSRT.py

Ce fichier contient uniquement sa fonction principale

1. Fonction del_vof_srt(st_dir, series)

```
def del_vof_srt(st_dir, series):
    for serie in series:
        serie_path = os.path.join(st_dir, serie)
        for root, dirs, files in os.walk(serie_path):
            for file in files:
                if file in ['vo.srt', 'vf.srt']:
                    file_path = os.path.join(root, file)
                    os.remove(file_path)
                    print(f"Deleted {file_path}")
```

↳ Supprimer tous les fichiers sous-titres qui ont pour nom vo.srt ou vf.srt

F) Fichier libRecode.py

1. Fonction recode(contenu)

```
def recode(contenu):
    # Remplace les Ã© par des é
    contenu = contenu.replace('Ã©', 'é')
    # Remplace les Ã¨ par des è
    contenu = contenu.replace('Ã¨', 'è')
    # Remplace les Ãª par des ê
    contenu = contenu.replace('Ãª', 'ê')
    ...
    # Remplace à‹ par Ë
    contenu = contenu.replace('à‹', 'Ë')
    print("Recode terminé")
    return contenu
```

↳ Corriger les caractères erronés en les remplaçant par les bons

G) Fichier libSrt.py

Le script de ce fichier est utile quand il faut manipuler des fichiers qui ont l'extension srt.

H) Fichier 0_main.py

Le fichier python 0_main.py récupère les fichiers de sous-titres bruts dans l'attribut nommé "st_dir" et lance les fonctions principales des fichiers python vu précédemment. La fonction appelée "del_journeyman(st_dir)" permet de gérer une exception du cas du dossier de la série Journeyman en le supprimant.

```python
st_dir = 'sous-titres'

def main():
    start_time = time.time()
    a_unzip.unzip(st_dir)
    del_journeyman(st_dir)
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    a_unzip.clean_folder_name(st_dir)
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    b_clean.clean(st_dir, series)
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    c_combine.combine(st_dir, series)
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    d_clean_VO_VF.clean_VO_VF(st_dir, series)
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    del_vofSRT.del_vof_srt(st_dir, series)
    end_time = time.time()
    print("Fin du traitement en", round(end_time - start_time, 2), "secondes")
```

III - Sous titres (données)

Avant le nettoyage, les fichiers de sous-titres étaient dispersés dans des archives compressées, rendant l'organisation du projet peu claire et difficile à gérer. Chaque série avait ses fichiers de sous-titres répartis dans plusieurs endroits, ce qui compliquait l'exploitation des données.

Après avoir effectué le processus de nettoyage, les fichiers ont été réorganisés de manière plus structurée. Désormais, chaque série possède son propre dossier sous le répertoire principal "sous-titres". À l'intérieur de chaque dossier de série, on trouve deux fichiers spécifiques : vo.txt pour la version originale (VO) et vf.txt pour la version française (VF). Cette réorganisation simplifie l'accès aux sous-titres et permet une gestion plus claire et plus efficace des données.

IV - Framework Flask

Dans cette partie du projet, nous avons utilisé le framework Flask pour développer l'application web. Flask structure l'application de manière à séparer les différentes fonctionnalités en dossiers spécifiques :

- api : Contient la logique métier de l'application, en particulier les fonctions pour interagir avec des APIs externes, comme la récupération des informations sur les séries.
- static : Regroupe les fichiers statiques tels que les fichiers JavaScript et CSS qui sont directement envoyés au client sans modification.
- templates : Contient les fichiers HTML (modèles) utilisés pour générer dynamiquement les pages web envoyées à l'utilisateur, en y insérant des données provenant du backend.
- manage.py : Fichier principal pour gérer l'exécution de l'application, souvent utilisé pour définir les routes, démarrer le serveur ou gérer des tâches liées à l'administration de l'application Flask.

Cette organisation permet une séparation claire entre la logique de traitement, la gestion des ressources statiques et la présentation dynamique des pages web.

A) Dossier api

1. Fichier .env

Les variables d'environnement sont des mécanismes puissants pour gérer les configurations des projets de développement de manière flexible et sécurisée. Elles stockent des informations sous forme de paires clé-valeur, accessibles par des langages comme Python, tout en protégeant ces données sensibles contre tout accès non autorisé.

Dans notre cas, ce fichier stocke la clé d'une API. Cette clé est utilisée pour authentifier les requêtes envoyées à l'API OMDB, qui fournit des informations sur les séries et les films.

```
OMDB_API_KEY=ee69e4e2
```

2. Fichier api.py

2.1 Chargement des variables d'environnements

La ligne load_dotenv() lit le contenu du fichier .env et charge les variables dans l'environnement système du programme.
La clé est récupérée à l'aide de os.getenv('OMDB_API_KEY'), ce qui retourne la valeur associée à OMDB_API_KEY dans le fichier .env. Cette clé est ensuite stockée dans la variable OMDB_API_KEY.

```
load_dotenv()
OMDB_API_KEY = os.getenv('OMDB_API_KEY')
```

2.2 Chemin vers le répertoire des sous-titres

Cette ligne de code attribue la chaîne de caractères 'sous-titres' à la variable st_dir. Cette variable représente le chemin vers un répertoire nommé "sous-titres" et est utilisée dans le code pour accéder et manipuler les fichiers texte des sous-titres.

```
st_dir = 'sous-titres'
```

2.3 Fonction get_info_serie(title)

```
def get_info_serie(title):
    url = f'http://www.omdbapi.com/?apikey={OMDB_API_KEY}&t={title}'
    response = requests.get(url)
    response.raise_for_status()
    return response.json()
```

↳ Utiliser la clef API OMDB pour construire l'URL permettant d'interroger l'API (cela est nécessaire pour que l'API accepte et traite la requête
↳ Récupérer les informations concernant une série en fonction de son titre en envoyant une requête à l'API OMDB
↳ Retourne un dictionnaire python

2.4 Fonction get_serie_problem_name(dir)

```
def get_serie_problem_name(dir):
    series = [folder for folder in os.listdir(dir) if os.path.isdir(os.path.join(dir, folder))]
    for serie in series:
        serie_info = get_info_serie(serie)
        if not 'Title' in serie_info:
            print(serie)
```

↳ Utilise la fonction get_info_serie(title) du fichier api.py
↳ Valider ou corriger les noms de tous les dossiers présents dans le répertoire st_dir pour que ces noms soit reconnus par l'API OMDB. Ces noms de dossier correspondent aux noms des séries.

2.5 Fonction get_top5(mots)

```python
def get_top5(mots):
    top5 = tf_idf.main(st_dir, mots)
    return top5
```

↳ Utilise la fonction main(st_dir, mot_input) du fichier tf_idf.py
↳ Obtenir le nom des cinq séries les plus pertinentes en fonction de mots-clefs.
↳ Retourne une liste python

2.6 Fonction get_top5_json(mots)

```python
def get_top5_json(mots):
    top5 = tf_idf.main(st_dir, mots)
    if not top5:
        return {"error": "Aucune série trouvée"}

    top5_json = []
    for serie in top5:
        try:
            serie_info = get_info_serie(serie)
            if not serie_info:
                print(f"No data found for {serie}")
                continue
            top5_json.append(serie_info)
        except requests.RequestException as e:
            print(f"Error fetching data for {serie}: {e}")
            continue
    return top5_json
```

↳ Utilise la fonction get_info_serie(title) du fichier api.py
↳ Utilise la fonction main(st_dir, mot_input) du fichier tf_idf.py
↳ Obtenir le nom et les informations des cinq séries les plus pertinentes en fonction de mots-clefs.
↳ Retourne une liste au format JSON

2.7 Fonction get_all_serie_json(st_dir)

```python
def get_all_serie_json(st_dir):
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    series_json = []
    for serie in series:
        try:
            serie_info = get_info_serie(serie)
            if not serie_info:
                print(f"No data found for {serie}")
                continue
            series_json.append(serie_info)
        except requests.RequestException as e:
            print(f"Error fetching data for {serie}: {e}")
            continue
    return series_json
```

↳ Utilise la fonction get_info_serie(title) du fichier api.py
↳ Obtenir le nom et les informations de toutes les séries présentes dans le répertoire st_dir
↳ Retourne une liste au format JSON

2.8 Fonction get_all_serie_json_by5(st_dir, page)

Par manque de temps, cette fonction n'a pas été utilisée dans ce projet mais elle aurait pu apporter plus de légèreté à la page du site qui a pour nom "AllSeries".

```python
def get_all_serie_json_by5(st_dir, page):
    series = [folder for folder in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, folder))]
    series_json = []
    for i in range((page - 1) * 5, min(page * 5, len(series))):
        serie = series[i]
        try:
            serie_info = get_info_serie(serie)
            if not serie_info:
                print(f"No data found for {serie}")
                continue
            series_json.append(serie_info)
        except requests.RequestException as e:
            print(f"Error fetching data for {serie}: {e}")
            continue
    return series_json
```

↳ Utilise la fonction get_info_serie(title) du fichier api.py
↳ Obtenir le nom et les informations de toutes les séries présentes dans le répertoire st_dir
↳ Retourne une liste de dictionnaire python, chaque dictionnaire contenant les détails d'une série

3. Fichier tf_idf.py

3.1 Fonction construct_series(st_dir)

```
def construct_series(st_dir):
    return {
        serie: [episode for episode in os.listdir(os.path.join(st_dir, serie)) if episode.endswith(".txt")]
        for serie in os.listdir(st_dir) if os.path.isdir(os.path.join(st_dir, serie))
    }
```

↳ Construire un dictionnaire où les clés sont les noms des séries (sous-dossiers présents dans le répertoire st_dir) et les valeurs sont des listes contenant les noms des fichiers textes présents dans ces sous-dossiers.
↳ Retourne ce dictionnaire python

3.2 Fonction construct_series_recommandation(st_dir, liked_series, disliked_series)

```
def construct_series_recommandation(st_dir, liked_series, disliked_series):
    return {
        serie: [episode for episode in os.listdir(os.path.join(st_dir, serie)) if episode.endswith(".txt")]
        for serie in os.listdir(st_dir)
        if os.path.isdir(os.path.join(st_dir, serie)) and serie not in liked_series and serie not in disliked_series
    }
```

↳ Variante de la fonction construct_series(st-dir) du fichier tf_idf.py mais exclut les sous-dossiers du répertoire st_dir qui ont les mêmes noms que le noms des séries aimées et non aimées
↳ Retourne un dictionnaire python

3.3 Fonction load_documents(st_dir, series)

```
def load_documents(st_dir, series):
    documents = []
    for serie, episodes in series.items():
        for episode in episodes:
            with open(os.path.join(st_dir, serie, episode), 'r', encoding='latin-1') as file:
                documents.append(file.read())
    return documents
```

↳ Parcourir le dictionnaire avec les séries
↳ Ouvrir chaque fichier texte et charger son contenu dans une liste appelée “documents”
↳ Retourne une liste python

3.3 Fonction process_documents(documents)

Cette fonction prétraite les documents texte en utilisant la méthode TF-IDF (Term Frequency - Inverse Document Frequency).

```
def process_documents(documents):
    vectorizer = TfidfVectorizer(stop_words=None, min_df=0.01, max_df=0.95)
    tfidf_matrix = vectorizer.fit_transform(documents)

    non_zero_columns = tfidf_matrix.max(axis=0).toarray().flatten() > 0
    filtered_tfidf_matrix = tfidf_matrix[:, non_zero_columns]

    filtered_feature_names = np.array(vectorizer.get_feature_names_out())[non_zero_columns]

    filtered_vectorizer = TfidfVectorizer(vocabulary=filtered_feature_names)
    filtered_vectorizer.fit(documents)

    return filtered_tfidf_matrix, filtered_vectorizer
```

↳ TfidfVectorizer est un outil scikit-learn qui convertit un ensemble de documents texte en une matrice de termes. Chacun de ces termes est pondéré en fonction de sa fréquence (TF) et de sa rareté (IDF)
↳ L'option stop_words=None indique que tous les mots courants tels que "le", "la", "les", seront pris en compte. Aucun mot n'est filtré par une liste de mots vides (stop words).
↳ L'option "min_df=0.01" indique que pour qu'un mot soit pris en compte, il doit composer au moins 1% du document texte. Ainsi les mots trop rares sont ignorés
↳ L'option "max_df=0.95" indique que pour qu'un mot soit pris en compte, il doit composer au maximum 95% du document texte. Ainsi les mots trop fréquents sont ignorés
↳ La deuxième ligne du code génère une matrice TF-IDF, où chaque ligne représente un document et chaque colonne représente un terme pondéré.
↳ Les termes dont le poids maximum est 0 dans la matrice (c'est-à-dire ceux qui n'ont aucun impact significatif) sont ensuite filtrés à l'aide de la condition de la troisième ligne. Cela permet d'éliminer les termes inutiles.
↳ Les mots correspondants aux colonnes restantes (après filtrage) sont extraits à l'aide de "vectorizer.get_feature_names_out()".
↳ Un nouveau TfidfVectorizer, limité au vocabulaire filtré, est créé. Celui-ci est ajusté aux documents pour garantir que seuls les termes pertinents sont considérés.
↳ Retourne la matrice TF-IDF filtrée "filtered_tfidf_matrix", qui ne contient que des colonnes correspondant aux termes retenus.
↳ Retourne le vectoriseur filtré "filtered_vectorizer", qui peut être utilisé pour transformer de nouveaux documents ou explorer le vocabulaire retenu.

3.4 Fonction find_most_similar_series(mot_input, vectorizer, tfidf_matrix, series, documents)

```python
def find_most_similar_series(mot_input, vectorizer, tfidf_matrix, series, documents):
    input_vector = vectorizer.transform([" ".join(mot_input)])
    similarities = cosine_similarity(input_vector, tfidf_matrix)
    if similarities.size == 0:
        return None

    top_indices = np.argsort(similarities[0])[::-1][:5]
    top_series = []
    for index in top_indices:
        serie_name = list(series.keys())[index // len(series[list(series.keys())[0]])]
        top_series.append(serie_name)
    return top_series
```

↳ Transformer les mots clefs (mot_input) en un vecteur TF-IDF
↳ Calculer les similarités cosinus entre ce vecteur et la matrice TF-IDF des documents texte
↳ Identifier les séries les plus similaires
↳ Retourne une liste de cinq noms des séries les plus pertinentes

3.5 Fonction main(st_dir, mot_input)

```python
def main(st_dir, mot_input):
    print("Chargement des sous-titres...")
    series = construct_series(st_dir)
    print("Chargement des documents...")
    documents = load_documents(st_dir, series)

    vectorizer_path = 'vectorizer.pkl'
    tfidf_matrix_path = 'tfidf_matrix.pkl'

    if os.path.exists(vectorizer_path) and os.path.exists(tfidf_matrix_path):
        print("Fichier trouvé.")
        with open(vectorizer_path, 'rb') as file:
            vectorizer = pickle.load(file)
        with open(tfidf_matrix_path, 'rb') as file:
            tfidf_matrix = pickle.load(file)
    else:
        print("Fichier non trouvé.")
        print("Calcul de la matrice tfidf et du vectorizer...")
        tfidf_matrix, vectorizer = process_documents(documents)
        print("Enregistrement de la matrice tfidf et du vectorizer...")
        with open(tfidf_matrix_path, 'wb') as file:
            pickle.dump(tfidf_matrix, file)
        with open(vectorizer_path, 'wb') as file:
            pickle.dump(vectorizer, file)

    print("Recherche de la série la plus similaire...")
    return find_most_similar_series(mot_input, vectorizer, tfidf_matrix, series, documents)
```

↳ Utiliser la fonction construct_series(st_dir) du fichier tf_idf.py
↳ Utiliser la fonction load_documents(st_dir, series) du fichier tf_idf.py
↳ Vérifier si des fichiers contenant une matrice TF-IDF et son vectorizer existent déjà en mémoire (fichiers pickle). Si ce n'est pas le cas, utilisation de la fonction process_documents(documents)
↳ Sauvegarder les résultats de la fonction process_documents. Ces résultats seront utilisés dans pour un usage ultérieur, cela permettra de gagner du temps pour la prochaine exécution.
↳ Utiliser la fonction find_most_similar_series(mot_input, vectorizer, tfidf_matrix, series, documents) du fichier tf_idf.py
↳ Retourne la liste des cinq séries les plus pertinentes à renvoyer en fonction de plusieurs mots-clefs

3.6 Fonction get_recommandation(st_dir)

```python
def get_recommandation(st_dir):
    series_like = session.get('liked', [])
    series_dislike = session.get('disliked', [])

    series = construct_series_recommandation(st_dir, series_like, series_dislike)
    documents = load_documents(st_dir, series)

    print("Calcul de la matrice tfidf...")
    tfidf_matrix, vectorizer = process_documents(documents)

    total_similarities = np.zeros(tfidf_matrix.shape[0])

    for serie_like in series_like:
        serie_like = str(serie_like)
        input_vector = vectorizer.transform([serie_like])
        similarities = cosine_similarity(input_vector, tfidf_matrix)
        total_similarities += similarities[0]

    sorted_indices = np.argsort(total_similarities)[::-1]
    max_indices = min(len(series), len(sorted_indices))

    recommandation = []
    for i in sorted_indices[:max_indices]:
        if i < len(series):
            recommandation.append(list(series.keys())[i])

    return recommandation[:5]
```

↳ Récupérer les séries aimées et non aimées depuis la session Flask
↳ Utiliser la fonction construct_series_recommandation(st_dir, liked_series, disliked_series) du fichier tf_idf.py
↳ Utiliser la fonction load_documents(st_dir, series) du fichier tf_idf.py
↳ Utiliser la fonction process_documents(documents) du fichier tf_idf.py
↳ Calculer les similitudes entre les séries aimées et les séries non évaluées
↳ Trier les séries non évaluées par leur pertinence dans l'ordre décroissant "[::-1]". Cela permet d'obtenir les séries les plus pertinentes en priorité.
↳ Retourne une liste de cinq noms de séries recommandées en fonctions de leurs places de pertinence (les cinq premières).
↳ Attention cette fonction marche mais est incomplète, car elle se base uniquement sur les séries "aimées" pour effectuer l'analyse, sans prendre en compte les séries "non aimées". Cela peut entraîner des suggestions biaisées qui ne reflètent pas pleinement les préférences de l'utilisateur.

3.7 Fonction getLike() et Fonction getDislike()

```python
def getLike():
    return session.get('liked', [])

def getDislike():
    return session.get('disliked', [])
```

↳ Retourne les noms des séries aimées ou non aimées sous forme de liste python.

B) Dossier static

Les script des trois fichiers javascript ont quelques similarités. Ils ne s'exécutent qu'une fois le DOM de la page prêt. Chaque script permet également l'envoi d'un message à l'url "/feedback" dans le fichier manage.py quand une série est aimée ou non aimée. Il y a également une fonction qui est la même dans ces trois fichiers qui permet de récupérer le titre des séries aimées et non aimées par l'utilisateur évaluées précédemment et de mettre à jour visuellement les boutons "J'aime" ou "Je n'aime pas" sur l'interface du site par séries. Il s'agit de la fonction updateButtons(). Nous allons maintenant voir ce qu'il y a en plus dans ces fichiers.

1. Fichier allSeries.js et recommandation.js

Ces fichiers utilisent la fonction commune updateButtons() pour synchroniser l'état visuel des boutons "J'aime" ou "Je n'aime pas" sur l'interface en fonction des préférences déjà enregistrées de l'utilisateur. Ils incluent la gestion du fonctionnement des clics sur les boutons "J'aime" ou "Je n'aime pas".

2. Fichier index.js

Ce qui est important dans ce fichier c'est qu'il permet de créer une fonction qui exécute la recherche de séries par mots-clefs préalablement donnés par l'utilisateur et qui retourne un résultat. Cette fonction s'appelle searchSeries().

Voici les étapes de fonctionnement de cette fonction :
↳ valider la saisie utilisateur (récupère les mots-clefs et vérifie que le nombre de ces mots est compris entre trois et dix)
↳ Effectuer une requête Ajax à l'URL "/search" dans le fichier manage.py
↳ Effacer l'ancien contenu de l'élément HTML "resultList" qui contient le résultat des séries recherchées. Cela permet au nouveau résultat de remplacer les anciens.
↳ Vérifier qu'il n'y a pas de problème ou d'erreur au niveau du serveur.
↳ Remplir l'élément HTML "resultList" avec le résultat de la recherche par mot-clefs
↳ Mettre à jour les boutons d'évaluation en faisant appel à la fonction updateButtons()

Le fichier index.js gère également le fonctionnement des clics sur les boutons “J’aime”, “Je n’aime pas” et “Rechercher”. Il active l’évènement lancé après que l’utilisateur ait appuyé sur la touche entrée de son clavier (touche 13).

3. Fichier styles.css

Le fichier style.css gère l’apparence et la mise en forme de l’interface pour toutes les pages HTML du site, en harmonisant les couleurs, les polices et la disposition des éléments. Dans le cadre du projet, nous avons également commencé à travailler sur la responsivité du site pour qu’il s’adapte aux différents formats d’écran (ordinateurs, tablettes, téléphones). Cependant, cette fonctionnalité n’est pas encore entièrement implémentée.

C) Dossier templates

Les fichiers HTML présents dans le dossier **templates** de Flask définissent la structure et le contenu visuel des pages de l'application web.
Les trois fichiers qui vont suivre incluent trois choses importantes. Ils incluent tous le fichier javascript qui leur correspond, la bibliothèque qui permet l’utilisation des boutons d’évaluation, et la bibliothèque JQuery qui simplifie les interactions avec le DOM (Document Object Model), la gestion des événements, les animations et les requêtes AJAX dans les pages web.

1. Fichier allSeries.html

↳ Interface de la page AllSeries : affiche les boutons de navigation du site ainsi que la liste de toutes les séries connus par les site web ainsi que leurs informations.
↳ Utilisation de Jinja2 pour générer une liste HTML dynamique en fonction des données des séries, incluant affiche, titre, genre, intrigue, acteurs, scénaristes, réalisateur, et récompenses.
↳ Intégration de boutons "J’aime" et "Je n’aime pas", dont l'état (classe et couleur) est géré dynamiquement via Jinja2 en fonction des listes liked et disliked.
↳ Structure des données compatible avec allSeries.js pour gérer les clics sur les boutons et synchroniser les préférences via la fonction updateButtons().

2. Fichier index.html

↳ Interface de la page d’accueil du site “RecoSeries” (page de recherche de séries par mots clefs) : affiche les boutons de navigation du site ainsi que la liste des séries résultantes de la recherche par mots clefs par l’utilisateur, accompagnées de leurs informations détaillées.
↳ Fournir une interface de recherche où l'utilisateur peut entrer des mots-clés
↳ Afficher les résultats de la recherche dans une liste sous le champ de recherche.

3. Fichier recommandation.html

↳ Interface de la page AllSeries : affiche les boutons de navigation du site ainsi que la liste des séries recommandées, basée sur les évaluations données par l’utilisateur, accompagnées de leurs informations détaillées.

↳ Utilisation de Jinja2 pour générer une liste HTML dynamique basée sur les données des séries recommandées, incluant les affiche, titre, genre, intrigue, acteurs, scénaristes, réalisateur, et récompenses.
↳ Les boutons "J'aime" et "Je n'aime pas" sont affichés avec des classes et couleurs modifiées dynamiquement grâce à Jinja2, selon si une série est déjà aimée ou non par l'utilisateur.
↳ Le code HTML est structuré de manière à faciliter l'interaction avec le fichier recommandation.js, qui gère les clics sur les boutons et la synchronisation de l'état des préférences.

D) Fichier manage.py

Ce fichier est un script Python qui utilise le framework Flask pour créer une application web interactive et dynamique. Il assure la gestion des routes, la configuration de l'application, et la logique côté serveur. Voici les principales actions qu'il effectue dans notre cas :

↳ Importation du module Flask et importation du module personnalisé des fonctions du dossier Flask api
↳ Définition d'une clef secrète qui signe les cookies de session et la configuration de ces mêmes cookies
↳ Récupération du chemin d'accès au nom du répertoire avec les sous-titres
↳ Initialisation des sessions avec "@before_request" (action exécutée avant chaque requête qui initialise les deux listes : liked et disliked)

De plus, dans le fichier manage.py, des routes définissent les URL accessibles de l'application web et relient ces URL à des fonctions spécifiques. Chaque route est associée à une fonction Python qui détermine la réponse à envoyer au navigateur (par exemple, une page HTML ou des données JSON). Les routes sont créées à l'aide de décorateurs Flask (comme @app.route) et permettent de structurer la logique de l'application en gérant les requêtes entrantes et leurs réponses associées.

Routes / Requêtes	Méthodes HTTP	Rôle
/	GET	Affiche la page d'accueil
/recommandation	GET	Affiche les recommandations basés sur les "J'aime"
/all_series	GET	Afficher une liste paginée de toutes les séries dsponibles
/search	GET	Recherche des séries avec des mots-clefs
/getlikes	GET	Renvoie la liste des séries aimées
/getdislikes	GET	Renvoie la liste des séries non aimées
/feedback	POST	Enregistre l'évaluation "J'aime" ou "Je n'aimes pas" pour une série

V - Docker

A) Docker

Docker est une plateforme installée sur une machine locale qui gère ce que l'on appelle des " conteneurs". Un conteneur va pourvoir empaqueter une application de manière isolée ce qui empêchera les problèmes de compatibilité matériel ou logiciel avec l'environnement final ou l'environnement d'un collègue dans un projet collaboratif.
Docker déploie des conteneurs à partir d'images Docker. Ces images disponibles et répertoriées sur une plateforme appelée Docker Hub contiennent des configurations préexistantes. Cependant il arrive souvent que l'on veuille que ces images soient personnalisées. C'est à ce moment-là qu'intervient le Dockerfile, un fichier semblable au fonctionnement d'une recette de cuisine.

B) Fichier Dockerfile

Ce fichier est un script qui contient une série d'instructions pour créer une nouvelle image Docker. Par exemple, il spécifie l'image de base (via l'instruction FROM) que l'on va utiliser pour construire cette nouvelle image. Dans notre projet l'image de base utilisée sera : python:3.10. Ce fichier spécifie également une suite de commandes nécessaires pour installer et configurer tout ce dont on a besoin pour le bon fonctionnement de l'environnement de notre application. Cela permet ensuite de déployer un conteneur sur mesure, parfaitement adapté aux besoins du projet. Dans le cas de notre projet, le Dockerfile définit ce que le docker doit faire quand le conteneur démarre : il faut démarrer un serveur web Flask en exécutant le fichier manage.py.

C) Fichier docker-compose.yml

Un Dockerfile fonctionne souvent de pair avec un fichier appelé docker-compose.yml, qui décrit et explique précisément à Docker le fonctionnement de l'ensemble d'un ou plusieurs conteneurs. Ce fichier est essentiel lorsqu'un projet nécessite plusieurs services Docker travaillant ensemble, ou lorsqu'il faut fournir des configurations détaillées pour un conteneur.
Le fichier docker-compose.yml de notre projet définit un seul service (conteneur) : le service web. Ce fichier demande à Docker Compose de construire une image Docker en utilisant le fichier Dockerfile qui dans notre cas se trouve dans le répertoire actuel. Ce fichier yml précise plusieurs informations importantes, telles que :

- le nom du conteneur ("web_container" dans notre cas)
- une politique de redémarrage automatique pour relancer le conteneur en cas d'échec ou d'arrêt.
- le mappage des ports, ici le port 8000 de l'hôte (la machine locale) est relié au port 8000 du conteneur. Cela signifie que le service web sera accessible à l'adresse suivante : http://127.0.0.1:8000.

Ainsi, grâce à Docker et ses outils, notre projet bénéficie d'un environnement stable, reproductible et prêt à être déployé efficacement.

VI - Autres fichiers du projet

A) Fichier .gitignore

Dans le cadre de ce projet, nous avons utilisé Git ainsi que GitHub, une plateforme en ligne qui permet d'héberger des projets. GitHub nous a permis de collaborer efficacement en équipe, de garder une trace de toutes les modifications apportées au code et de revenir à des versions antérieures en cas de besoin. De plus, Git nous a aidés à organiser le projet de manière structurée, à travailler sur différentes branches et à fusionner les contributions de manière cohérente.

Ce fichier permet de spécifier à Git (quand on l'utilise) qu'il n'est pas utile de transférer les fichiers du projet qui ont certaines extensions.

B) Fichier README.md

Un fichier README, généralement placé à la racine d'un projet, fournit des informations essentielles sur le système, le projet ou le logiciel. Il sert différents objectifs selon les utilisateurs : pour l'utilisateur final, il explique l'installation, la mise à jour ou l'utilisation; pour le développeur, il peut définir les lignes directrices du projet ou aider à reprendre un projet mis en pause; et pour d'autres développeurs, il précise les règles et fournit des instructions pour continuer le développement d'un projet.

C) Fichier requirements.txt

Le fichier requirements.txt est utilisé pour définir les bibliothèques Python nécessaires à ce projet. Ce fichier sera utilisé dans le Dockerfile pour installer automatiquement ces dépendances et garantir un environnement cohérent.

Bibliothèque	Utilité
Flask	Framework léger pour créer l'application web en Python.
python-dotenv	Permet de gérer les variables d'environnement via un fichier .env
scikit-learn	Permet de convertir des textes en vecteurs et de mesurer leur similarité.
numpy	Bibliothèque pour le calcul numérique, manipulant efficacement les tableaux et matrices multidimensionnels.
nltk	Outils pour le traitement du langage naturel
regex	Fournit des outils puissants pour le traitement et la manipulation de chaînes à l'aide d'expressions régulières.
chardet	Détecte automatiquement l'encodage des fichiers texte.

ftfy	Corrige les textes mal encodés ou les caractères corrompus.
wordninja	Sépare les mots collés dans des chaînes de caractères, utile pour le nettoyage de texte.
requests	Facilite les requêtes HTTP pour communiquer avec des APIs ou des services web externes.

D) Fichiers Pickle

Dans le cadre de ce projet, deux fichiers Pickle sont générés pour optimiser les calculs et faciliter la réutilisation des données prétraitées. Les fichiers Pickle sont des fichiers de stockage.

1. tfidf_matrix.pkl

Ce fichier contient la matrice TF-IDF, un ensemble de vecteurs numériques représentant les sous-titres des séries. Chaque vecteur encode les informations de fréquence des mots et leurs importances relatives (TF-IDF).

Il évite de recalculer les vecteurs TF-IDF pour les mêmes données, accélérant ainsi les processus d'analyse. Cette matrice est essentielle pour effectuer des comparaisons entre les mots-clés fournis par l'utilisateur et les sous-titres des séries afin de déterminer les correspondances les plus pertinentes.

Le matrice compressée est sauvegardée sous le format Pickle (.pkl), propre à Python, permettant de stocker des objets sérialisés. La matrice est donc manipulable via des bibliothèques Python comme “scikit-learn” et “numpy”.

2. vectorizer.pkl

Ce fichier contient l'objet TfidfVectorizer configuré et entraîné sur les sous-titres des séries. Il inclut notamment le vocabulaire et les paramètres utilisés pour convertir des documents en vecteurs TF-IDF.

Il permet de transformer efficacement de nouveaux textes ou mots-clés en vecteurs compatibles avec la matrice TF-IDF existante. En utilisant le même vectorizer, on garantit la cohérence entre les nouveaux vecteurs et ceux déjà calculés.

Comme tfidf_matrix.pkl, ce fichier est sauvegardé au format Pickle (.pkl), ce qui facilite son chargement et sa réutilisation dans les scripts Python.

VII - Glossaire

API (Application Programming Interface) : Interface permettant à deux systèmes logiciels de communiquer entre eux. Dans ce projet, l'API OMDB est utilisée pour récupérer des informations sur les séries.

Docker : Plateforme permettant de créer, déployer et exécuter des applications à l'intérieur de conteneurs isolés, assurant ainsi la portabilité et la compatibilité des applications.

Docker Compose : Outil pour définir et gérer plusieurs conteneurs Docker dans un environnement collaboratif ou multi-services.

Flask : Framework web minimaliste en Python utilisé pour développer des applications web dynamiques.

JSON (JavaScript Object Notation) : Format léger de structuration de données, utilisé ici pour transmettre des listes de séries ou leurs informations entre le frontend et le backend.
Pickle : Module Python permettant de sérialiser des objets en fichiers binaires pour les stocker ou les charger facilement, comme la matrice TF-IDF ou le vectoriseur.

TF-IDF (Term Frequency - Inverse Document Frequency) : Technique de pondération utilisée pour évaluer l'importance d'un mot dans un document par rapport à un corpus global. Dans ce projet, elle est appliquée aux sous-titres des séries.

TfidfVectorizer : Outil de la bibliothèque scikit-learn permettant de transformer un ensemble de documents en une matrice TF-IDF, utilisée pour analyser les sous-titres.

Nettoyage des données : Processus visant à structurer et purifier les fichiers bruts (sous-titres) en supprimant les éléments inutiles ou incohérents, essentiel pour assurer des analyses précises.

Session Flask : Système de stockage temporaire d'informations spécifiques à un utilisateur, telles que les séries "aimées" ou "non aimées", pendant sa session active.

Recommandation : Fonctionnalité permettant de suggérer des séries à un utilisateur en se basant sur ses préférences (séries aimées ou non évaluées).

Jinja2 : Moteur de template intégré à Flask, utilisé pour générer dynamiquement des pages HTML avec des données du backend.

Frontend : Partie visible de l'application web (HTML, CSS, JavaScript) qui interagit directement avec l'utilisateur.

Backend : Partie serveur de l'application (Flask, Python) qui traite les données et génère les réponses envoyées au frontend.

Requête HTTP : Demande envoyée par le frontend au backend pour obtenir des données ou effectuer une action (exemple : évaluer une série ou rechercher des séries).

Vectorisation : Processus de conversion de documents texte en vecteurs numériques pour une analyse informatique, utilisé dans la génération de la matrice TF-IDF.