Rapport

- Organisateur de fichiers -

Encadré par :

Maurer Alexandre

Réalisé par :

Jellal Khaoula

Kably Malak

Kouhail Oumaima

Introduction :

Le projet **File's Organizer** évolue dans un contexte particulièrement dynamique, marqué par la rapide révolution numérique. Cette ère, caractérisée par une croissance exponentielle du volume de données, présente des défis substantiels liés à la prolifération de fichiers de formats divers, contribuant ainsi à une complexité souvent perçue dans le paysage informatique. Cette dynamique crée un environnement favorable au désordre sur les bureaux numériques, où la quête rapide et précise de documents essentiels devient une tâche ardue.

L'excès de données numériques génère des implications directes sur la productivité et l'expérience utilisateur. Les utilisateurs, désemparés devant un bureau encombré de fichiers, sont confrontés à des obstacles majeurs pour maintenir un ordre structuré. Ce désordre peut entrainer des pertes de temps considérables et une frustration croissante, soulignant ainsi la nécessité impérieuse de trouver des solutions novatrices pour orchestrer de manière efficiente ces vastes ensembles de données.

Ainsi, **File's Organizer** se positionne comme une réponse réfléchie à ces défis omniprésents. En offrant une approche systémique et inventive pour organiser les fichiers, le projet ambitionne de transcender les contraintes imposées par cette surcharge informationnelle. En orientant l'utilisateur vers une gestion plus intuitive de ses données numériques, **File's Organizer** aspire à être une réponse pertinente et efficace dans cet univers complexe de fichiers variés.

Problématique :

La problématique adressée par le projet **File's Organizer** découle de la réalité quotidienne partagée par de nombreux utilisateurs, confrontés à la gestion de plus en plus complexe et encombrée de leurs fichiers numériques. Dans un monde où la quantité d'informations augmente de manière fulgurante, le bureau numérique devient rapidement un lieu propice à la surabondance de données. Cette surabondance crée une situation ingénieuse où les utilisateurs, par manque de temps ou de motivation, accumulent des fichiers sans maintenir un ordre clair.

Le constat général est que la plupart d'entre nous sont suffisamment paresseux pour organiser méticuleusement nos dossiers. Cette paresse, combinée à la croissance exponentielle du volume de données, entraine une accumulation rapide de fichiers sans structure apparente. Ainsi, lorsqu'il devient nécessaire de retrouver un fichier spécifique, l'effort requis pour parcourir des dossiers en constante expansion devient une tâche difficile et chronophage.

La difficulté de recherche est exacerbée lorsque le nom du fichier est oublié, laissant les utilisateurs dans une impasse. Dans ces situations, l'utilisateur se trouve à espérer qu'il existe une solution pour effectuer une recherche basée sur le contexte ou le thème des fichiers plutôt que de dépendre uniquement du nom du fichier. La recherche basée sur le thème pourrait

simplifier le processus en affichant tous les fichiers liés à un sujet spécifique, rendant ainsi la recherche plus intuitive et moins laborieuse.

Cependant, cette problématique ne se limite pas seulement à la recherche basée sur le thème. Elle s'étend également à la nécessité d'organiser les fichiers en fonction de leur extension. Lorsqu'on se retrouve avec une multitude de fichiers de formats différents, chaque extension représente une catégorie distincte. Le défi supplémentaire réside dans la création d'une structure claire pour ces catégories, facilitant ainsi la localisation rapide des fichiers en fonction de leur type.

Le projet **File's Organizer** aborde ces problématiques de manière proactive en proposant une solution qui va au-delà de la simple organisation par thème. En comprenant les défis auxquels sont confrontés les utilisateurs en matière de gestion des fichiers, l'application vise à simplifier le processus de recherche et d'organisation des fichiers numériques. Elle offre une réponse à la fois pratique et ingénieuse pour résoudre les problèmes de désordre, de difficulté de recherche et d'organisation par extension, améliorant ainsi l'expérience globale de l'utilisateur dans la gestion de ses fichiers numériques.

Stratégies pour l'Optimisation de l'Organisation des Fichiers :

Pour résoudre la problématique évoquée, le projet **File's Organizer** a mis en œuvre des méthodes et algorithmes innovants visant à simplifier la gestion des fichiers numériques de manière efficace et intuitive.

Une des fonctionnalités clés de l'application repose sur l'extraction de texte à partir d'images, permettant ainsi aux utilisateurs de traiter non seulement les fichiers texte, mais également ceux intégrant du contenu visuel. Cette méthode utilise la technologie de reconnaissance optique de caractères (OCR) pour convertir le texte présent dans les images en données exploitables. Cela élargit considérablement la portée de l'application, offrant une solution complète pour la gestion de divers types de fichiers, y compris ceux provenant de captures d'écran, de photos, ou de documents scannés.

Dans le processus d'organisation des fichiers basé sur le thème, **File's Organizer** recourt à la technique du TF-IDF pour évaluer la pertinence d'un document par rapport à un ensemble de mots-clés associés à chaque thème. TF-IDF est une méthode fréquemment utilisée dans le domaine du traitement du langage naturel qui attribue un poids à chaque terme en fonction de sa fréquence dans un document particulier et de sa rareté dans l'ensemble du corpus.

Tout d'abord, le texte extrait des documents ainsi que les mots-clés liés à chaque thème sont prétraités pour éliminer les éléments indésirables tels que la ponctuation et les mots fréquents mais peu informatifs (stop words). Ensuite, ces données sont converties en une matrice TF-IDF à l'aide d'un vectoriseur. Cette matrice reflète l'importance relative de chaque mot dans le contexte du document et du corpus global.

La similarité cosinus est ensuite utilisée pour mesurer l'angle entre les vecteurs TF-IDF des mots-clés du thème et du texte du document. La similarité cosinus fournit un score compris

entre 0 et 1, où 0 indique une absence totale de similarité, et 1 représente une similarité parfaite. Plus précisément, le score est calculé en évaluant le cosinus de l'angle entre les deux vecteurs, fournissant une mesure de la proximité sémantique entre le document et le thème spécifique.

Ainsi, le score obtenu grâce à cette méthodologie reflète la correspondance entre le contenu du document et les mots-clés associés au thème, permettant une classification précise des fichiers en fonction de leur pertinence.

En parallèle, l'organisation par extension, une caractéristique distincte de l'application, est mise en œuvre à travers des algorithmes spécifiques. Ces algorithmes classent les fichiers en fonction de leurs extensions respectives, créant ainsi une hiérarchie claire dans la structure des dossiers. L'utilisateur peut choisir entre une organisation basée sur le thème ou par extension, offrant une flexibilité maximale pour répondre à ses préférences individuelles.

L'utilisation de ces méthodes avancées, combinée à une interface utilisateur conviviale, fait de **File's Organizer** une solution complète et puissante pour résoudre les défis liés à la gestion des fichiers numériques. Cette approche multidimensionnelle garantit une expérience utilisateur optimale en fournissant des outils intelligents pour l'organisation, la recherche et la gestion efficace de fichiers, créant ainsi un bureau numérique plus ordonné et agréable.

Défis Rencontrés et Solutions Apportées :

Dans la résolution de cette problématique, nous avons été confrontés à une problématique spécifique liée au traitement des images dépourvues de texte. Notre objectif initial était d'inclure les images dans le processus d'organisation en attribuant des thèmes en fonction du contenu visuel. Cependant, cette tâche s'est révélée complexe, même avec des approches basées sur l'apprentissage automatique.

L'un des obstacles majeurs était la difficulté à élaborer un algorithme capable d'identifier de manière fiable le contenu des images. Même avec des techniques avancées de machine Learning, la détermination du thème en se basant uniquement sur le visuel s'avérait problématique. Par exemple, si l'image représentait un dessin d'un avion, l'algorithme pouvait correctement identifier l'objet comme un avion, mais la confusion pouvait survenir quant au thème approprié. Dans ce cas, le dessin pouvait être associé au thème "Voyage" ou "Transport" plutôt qu'à "Art".

Face à cette complexité et pour maintenir la cohérence du système, nous avons décidé de restreindre le traitement des images uniquement à celles contenant du texte. Cette approche a permis de garantir une interprétation plus précise du contenu, en se basant sur les informations textuelles présentes dans l'image. Bien que cette limitation réduise la portée du traitement des images, elle assure une classification plus fiable et évite les ambigüités potentielles liées à l'interprétation visuelle seule.

Analyse et explication du code :

Pour une compréhension plus approfondie du fonctionnement de l'application et pour une référence visuelle directe, des captures d'écran du code pertinent ont été incluses dans les appendices.

Méthode organize_files :

La méthode ***organize_files()*** est le noyau de l'application, orchestrant le processus d'organisation des fichiers. En récupérant le chemin du dossier spécifié par l'utilisateur, elle procède à un tri précis selon les options sélectionnées. Si l'organisation par extension est choisie, elle classe les fichiers en fonction de leur type. Si l'option par thème est préférée, elle analyse le contenu des fichiers pour les regrouper suivant des catégories définies. Une fois cette opération réalisée, elle s'assure de nettoyer les dossiers devenus vides à la suite de l'organisation. Enfin, elle informe l'utilisateur du succès de l'opération via l'étiquette d'information. Dans les détails qui suivent, vous découvrirez une explication approfondie et plus technique du fonctionnement précis de cette méthode :

- **Récupération du chemin du dossier :** Cette étape utilise une variable, ***folder_path***, pour stocker le chemin du dossier spécifié par l'utilisateur.
- **Vérification du chemin du dossier :** La condition ***if not folder_path*** vérifie si aucun chemin de dossier n'a été spécifié. Si cette condition est vérifiée (si folder_path est vide ou non défini), la fonction se termine prématurément en retournant None, indiquant ainsi qu'aucune opération ne peut être effectuée sans un chemin de dossier valide.
- **Organisation par Extension :** Si l'utilisateur a choisi l'option de tri par extension, cette partie du code appelle la fonction ***organize_by_extension()***. Cette fonction est chargée de trier les fichiers en fonction de leur type (texte, image, vidéo, etc.) en les déplaçant dans des dossiers correspondants.
- **Organisation par Thème :** Lorsque l'option de tri par thème est sélectionnée par l'utilisateur, le programme extrait les thèmes spécifiés par l'utilisateur à partir de l'interface graphique. Ces thèmes sont entrés dans un champ dédié où l'utilisateur entre une liste de mots-clés séparés par des virgules.Ensuite, le code divise cette chaîne de thèmes en une liste de mots-clés individuels, éliminant les espaces inutiles autour de chaque mot-clé à l'aide de la méthode ***strip()***. Cette liste résultante, nommée ***user_entered_themes***, est passée comme argument à la méthode ***organize_by_theme()***.
- **Suppression des dossiers vides :** Après que les fichiers ont été organisés, cette fonction, ***remove_empty_folders()***, est invoquée pour maintenir la propreté du dossier. Elle parcourt le dossier pour identifier et supprimer les dossiers devenus vides à la suite de l'organisation des fichiers.
- **Mise à jour de l'étiquette d'information :** Enfin, l'interface utilisateur est mise à jour pour refléter l'état après l'organisation des fichiers. L'étiquette d'information est modifiée pour informer l'utilisateur que l'opération d'organisation des fichiers a été effectuée avec succès.

Organisation par extension :

La partie d'organisation par extension dans le code fourni vise à trier les fichiers dans un dossier en fonction de leurs extensions respectives. Voici comment cela fonctionne :

- **Sélection du dossier à organiser :** L'utilisateur sélectionne un dossier à organiser via l'interface graphique.
- **Liste de tous les fichiers :** Le programme liste tous les fichiers dans le dossier sélectionné, y compris ceux des sous-dossiers, en utilisant la fonction ***list_files***.
- **Boucle sur chaque fichier :** Pour chaque fichier trouvé dans la liste, le programme effectue les étapes suivantes :
    1. **Vérification de l'appartenance au Git :** Cette étape vise à détecter si le fichier est associé à Git. Cela se fait généralement en vérifiant la présence d'un dossier caché nommé***. git***. Si ce dossier est repéré dans le chemin du fichier, le fichier est identifié comme lié à Git et sera ignoré lors du processus d'organisation.
    2. **Obtention de l'extension du fichier :** Cette action consiste à extraire l'extension du fichier pour déterminer le type de fichier. Par exemple, pour le fichier ***"document.txt***", l'extension serait ***"txt"***.
    3. **Détermination du chemin de destination :** En fonction de l'extension, un dossier de destination est déterminé pour le fichier. Par exemple, les fichiers avec l'extension ***".txt"*** pourraient être dirigés vers un dossier spécifique nommé ***"TextFiles"***.
    4. **Vérification et création du dossier de destination :** Avant de déplacer le fichier, le code vérifie si le dossier de destination pour cette extension existe. S'il n'existe pas, il est créé pour accueillir les fichiers de ce type.
    5. **Gestion des noms de fichiers en double :** S'il existe déjà un fichier avec le même nom dans le dossier de destination, le code génère un nouveau nom pour éviter les conflits. Par exemple, s'il y a déjà un ***"document.txt"***, le code renommera le fichier en ***"document_1.txt"***.
    6. **Déplacement et renommage des fichiers :** Une fois le dossier de destination déterminé pour chaque fichier, le processus gère les éventuels doublons de noms en ajoutant un suffixe numérique au fichier si nécessaire. Ensuite, il effectue le déplacement du fichier vers le répertoire de destination en utilisant ***os.rename()***, garantissant ainsi une organisation sans conflits ni doublons dans les dossiers cibles.

Cela garantit que chaque fichier est correctement classé dans un dossier spécifique en fonction de son type, avec une gestion adéquate des conflits de noms pour assurer l'intégrité des données.

Organisation par theme:

Dans notre approche d'organisation par thème, la fonction ***organize_by_theme*** joue un rôle central en coordonnant l'ensemble du processus. Veuillez trouver le code correspondant dans la section des appendices.

Vue d'Ensemble de l'Algorithme d'Organisation par Thème :

L'algorithme d'organisation par thème est conçu pour simplifier la gestion de fichiers en les classant automatiquement dans des dossiers thématiques, basés sur le contenu des fichiers et les thèmes fournis par l'utilisateur. Voici déroulement logique de ce processus :

1. *Transformation des fichiers en texte :*

- Les fichiers du dossier spécifié sont parcourus.
- Chaque fichier est converti en texte à l'aide de la fonction file_to_text.

2. *Génération des Mots-Clés pour Chaque Thème :*

- Une liste de mots-clés est générée pour chaque thème. Ces mots-clés serviront de base pour l'évaluation de la similarité avec le contenu des fichiers.

3. *Notation des Documents pour Chaque Thème :*

- Chaque document est évalué par rapport à chaque thème en utilisant la fonction calculate_score, qui mesure la similarité entre le texte du document et les mots-clés associés à chaque thème.

4. *Déplacement des Fichiers dans des Dossiers Respectifs :*

- Les fichiers sont déplacés dans des dossiers thématiques en fonction des scores attribués.
- Un seuil est défini pour déterminer si un document est suffisamment lié à un thème pour être déplacé dans son dossier dédié.

Exploration Détaillée de l'Algorithme d'Organisation par Thème :

Dans cette section, nous plongerons dans les détails de chaque étape de l'algorithme, décrivant comment il transforme efficacement les fichiers, génère des mots-clés, évalue les documents par thème, et organise les fichiers dans des dossiers thématiques. Découvrons ensemble le processus étape par étape.

Étape 1 : Transformation des fichiers en texte

Dans cette première étape, notre algorithme parcourt tous les fichiers du dossier spécifié, puis utilise la fonction file_to_text pour les convertir en texte exploitable.

Code Expliqué :

La fonction file_to_text prend en paramètre le chemin d'un fichier et retourne le texte extrait. Voici comment elle fonctionne pour différentes extensions de fichiers :

- Fichiers Texte (.txt) :
  Le contenu du fichier texte est simplement lu à l'aide de **open** et **read**.

- Fichiers Word (.docx, .doc) :
  La bibliothèque **docx2txt** est utilisée pour extraire le texte de ces fichiers.

- Fichiers PDF (.pdf) :
  PyPDF2 est utilisé pour extraire le texte de chaque page du PDF.

- Fichiers Image (. jpg, jpeg, webp) :
  Tesseract OCR est employé pour extraire le texte à partir de l'image.

- Fichiers Vidéo (.mp4) :
  a. Le fichier vidéo est converti en fichier audio WAV (output_audio.wav).
  b. La fonction audio_to_text est ensuite appelée pour convertir l'audio en texte.

- Fichiers Audio (.mp3) :
  La fonction audio_to_text est directement utilisée pour convertir l'audio en texte.

Fonction audio_to_text :

Cette fonction est cruciale pour la conversion des fichiers audio (MP3) en texte compréhensible. Voici une description détaillée de chaque étape de cette fonction :

1. Conversion MP3 en WAV :
   - La fonction commence par convertir le fichier audio au format MP3 en un fichier audio WAV. Cela est effectué avec l'aide de la bibliothèque **pydub.**
   - Le fichier WAV est sauvegardé temporairement sous le nom "output_audio.wav".

2. Reconnaissance de la Parole (Speech-to-Text) :
   - En utilisant la bibliothèque **speech_recognition**, la fonction lit le fichier WAV et extrait les données audios.
   - Ces données audios sont ensuite transmises à l'API Google Speech Recognition pour effectuer la conversion de la parole en texte.

3. Gestion des Erreurs :
   - La fonction est encapsulée dans un bloc **try-except** pour gérer les erreurs potentielles.
   - En cas d'erreur liée au fichier non trouvé (**FileNotFoundError)**, un message explicatif est renvoyé.
   - Si une autre exception se produit pendant le processus de conversion, un message d'erreur détaillé est renvoyé.

4. Nettoyage des Fichiers Temporaires :

- Le fichier WAV temporaire est supprimé dans la clause **finally** pour éviter l'accumulation de fichiers inutiles.

La fonction file_to_text renvoie le texte extrait en minuscules pour assurer une cohérence dans l'analyse ultérieure. Ce processus polyvalent garantit une transformation homogène de divers formats de fichiers en texte exploitable.

Etape 2 : Génération des Mots-Clés pour Chaque Thème

Utilisation de la fonction generate_related_keywords . Cette fonction joue un rôle crucial dans l'analyse thématique en générant une liste de mots-clés associés à un thème spécifié. Voici une explication détaillée des étapes de cette fonction :

1. **Obtention des Synsets pour le Thème :**
   - La fonction commence par obtenir les synsets (ensembles de synonymes) associés au thème spécifié en utilisant **WordNet**.
   - Les synsets sont obtenus en convertissant le thème en minuscules et en utilisant la bibliothèque WordNet (**wordnet.synsets**).

2. **Extraction des Lemmes :**
   - Pour chaque synset obtenu, les lemmes (formes de base des mots) sont extraits et ajoutés à l'ensemble des mots-clés liés.
   - Cela inclut les lemmes du synset cible ainsi que ceux des **hypernyms** (termes plus généraux) et des **hyponyms** (termes plus spécifiques).

3. **Limitation du Nombre de Mots-Clés :**
   - La liste des mots-clés liés est tronquée pour ne conserver qu'un nombre spécifié (num_keywords) de mots-clés.
   - Cela permet de contrôler la taille de la liste de mots-clés générée, ce qui est particulièrement utile pour éviter une liste trop longue.

4. **Renvoi de la Liste de Mots-Clés :**
   - La fonction renvoie la liste finale des mots-clés liés au thème, prête à être utilisée dans le processus d'évaluation des documents.

Etape 3 : Notation des documents pour chaque thème

Utilisation de la fonction calculate_score qui est responsable de noter un document par rapport à un ensemble de mots-clés associés à un thème donné. Voici un aperçu du fonctionnement de cette fonction :

1. **Prétraitement du Texte :**
   Le texte du document est prétraité en utilisant la fonction preprocess_text. Ce prétraitement consiste en deux étapes principales :
   - *Suppression de la Ponctuation* : Les caractères de ponctuation sont éliminés du texte.
   - *Suppression des Mots Vides* : Les mots vides (stop words) couramment utilisés, tels que "and", "the", etc., sont retirés du texte.

2. **Combinaison du Texte Prétraité et des Mots-Clés :**
   Le texte prétraité est combiné avec la liste des mots-clés associés au thème dans une liste appelée text_and_keywords.

3. **Création d'un Vectoriseur TF-IDF :**
   Un vectoriseur **TF-IDF** (Term Frequency-Inverse Document Frequency) est créé à l'aide de la classe **TfidfVectorizer** de la bibliothèque **scikit-learn**. Cette étape est cruciale pour convertir le texte en une représentation numérique significative.

4. **Transformation du Texte et des Mots-Clés :**
   Le texte prétraité et les mots-clés sont ajustés et transformés à l'aide du vectoriseur **TF-IDF**, produisant ainsi une matrice TF-IDF.

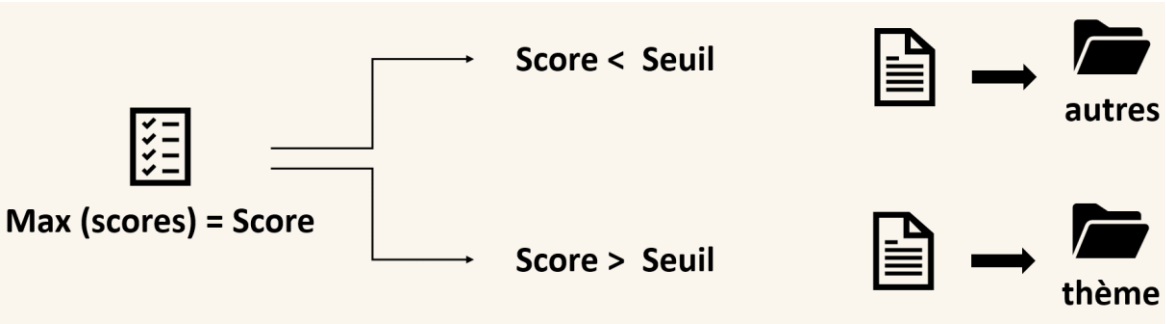
5. **Calcul de la Similarité Cosinus :**
   La similarité cosinus est calculée entre le texte prétraité et les mots-clés. La similarité cosinus mesure la similarité directionnelle entre deux vecteurs dans un espace vectoriel.

6. **Obtention du Score de Similarité :**
   Le score de similarité est extrait de la matrice de similarité. Dans ce contexte, le score se trouve à la position (0, 1) car 0 représente le texte prétraité et 1 représente les mots-clés.

7. **Renvoi du Score de Similarité :**
   Le score de similarité calculé est renvoyé en tant que résultat de la fonction, représentant la mesure de la similarité entre le texte du document et les mots-clés associés au thème.

Etape 4 : Déplacement des fichiers dans des dossiers respectifs

Au cours de la dernière étape, notre algorithme examine attentivement chaque document. Nous cherchons à identifier le score le plus élevé parmi tous les scores associés à chaque thème pour ce document particulier. En parallèle, nous évaluons ce score en le comparant au seuil que nous avons défini antérieurement. Ce seuil agit comme un critère de pertinence, déterminant si le document est suffisamment lié à un thème pour justifier son déplacement vers un dossier spécifique.



Logique de la Décision :

- ✓ Si le score est inférieur au seuil, cela suggère que le document ne présente pas une correspondance significative avec les thèmes prédéfinis. En conséquence, le fichier est redirigé vers le dossier "autres".
- ✓ En revanche, si le score est égal ou supérieur au seuil, nous concluons que le document est pertinent par rapport à un thème spécifique. Dans ce cas, le fichier est dirigé vers le dossier correspondant au thème avec le score le plus élevé.

Dans cette étape finale de l'algorithme d'organisation par thème, les fichiers sont déplacés vers des dossiers thématiques en fonction de leurs scores attribués. La fonction clé utilisée pour ce déplacement est move_file_to_folder. Voici une explication détaillée de cette fonction :

1. **Chemin du Dossier de Destination :**
   dest_folder est créé en combinant le répertoire du fichier source (os.path.dirname(filepath)) avec le nom du dossier de destination spécifié (folder_name).

2. **Création du Dossier de Destination :**
   os.makedirs est utilisé pour créer le dossier de destination s'il n'existe pas déjà (exist_ok=True).

3. **Gestion des Noms de Fichier en Double :**
   Pour éviter les conflits de noms de fichiers, la fonction vérifie si le fichier de destination existe déjà. Si c'est le cas, elle ajoute un numéro séquentiel au nom du fichier ({base_filename}_{counter}{file_extension}) jusqu'à ce qu'un nom de fichier unique soit trouvé.

4. **Déplacement du Fichier :**
   La fonction **shutil.move** est utilisée pour déplacer le fichier vers le dossier de destination final.

5. **Gestion des Erreurs :**
   La fonction est entourée d'une structure **try...except** pour gérer d'éventuelles erreurs lors du déplacement du fichier. Tout problème rencontré sera affiché dans la console.

Conclusion :

En somme, cette application se révèle être un outil efficace pour organiser avec précision les fichiers présents dans un dossier défini. Grâce à ses deux méthodes distinctes d'organisation par extension ou par thème, elle permet une classification intelligente des fichiers, facilitant ainsi leur recherche et leur accès ultérieur. En triant méthodiquement les fichiers dans des dossiers spécifiques et en veillant à la propreté de l'espace de stockage en supprimant les dossiers vides, cette application offre une gestion optimale des contenus. Son processus de déplacement et de renommage des fichiers assure une organisation structurée, évitant tout doublon ou conflit de noms. En définitive, cette application offre une solution pratique et efficace pour une gestion rationalisée des fichiers.

Appendices :

La fonction organize_files:

```python
def organize_files(self):
    folder_path = self.folder_path.text()

    if not folder_path:
        return

    if self.extension_radio.isChecked():
        self.organize_by_extension(folder_path)

    if self.theme_radio.isChecked():
        user_entered_themes = [
            theme.strip() for theme in self.current_theme.split(",")
        ]
        self.organize_by_theme(folder_path, user_entered_themes)

    # Remove empty folders after organizing files
    self.remove_empty_folders(folder_path)

    self.info_label.setText(f"Files in {folder_path} organized!")
```

La fonction organize_by_extension():

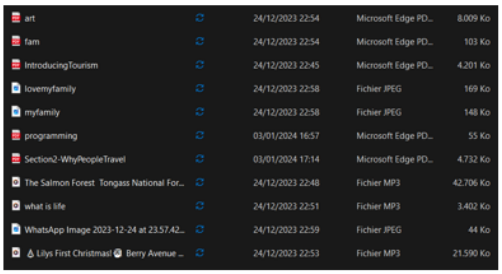
```python
def organize_by_extension(self, folder_path):
    all_files = self.list_files(folder_path)

    for file_path in all_files:
        if ".git" in file_path:
            continue

        extension_start = file_path.rfind(".")
        filename_start = file_path.rfind(os.sep) + 1


        if extension_start != -1:
            file_extension = file_path[extension_start + 1 :].lower()
        else:
            file_extension = "no_extension"

        filename = file_path[filename_start:]
        dest_folder = os.path.join(folder_path, file_extension)

        if not os.path.exists(dest_folder):
            os.mkdir(dest_folder)

        dest_file_path = os.path.join(dest_folder, filename)
        base_filename = filename[: extension_start - filename_start]
        counter = 1
        while os.path.exists(dest_file_path):
            dest_file_path = os.path.join(
                dest_folder, f"{base_filename}_{counter}.{file_extension}"
            )
            counter += 1

        os.rename(file_path, dest_file_path)
```

La fonction organize_by_theme():

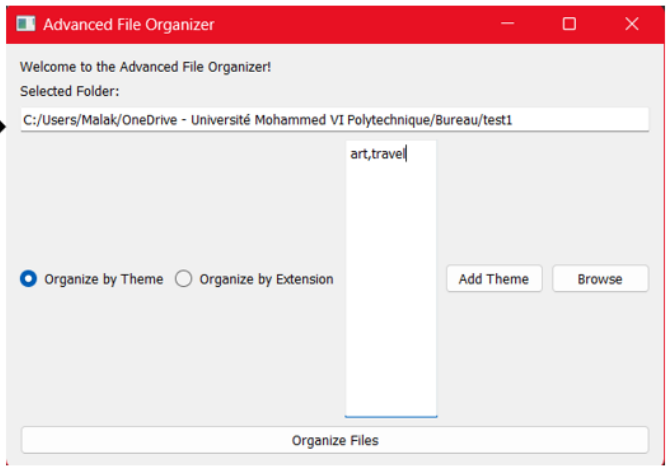
```python
def organize_by_theme(self, folder_path, themes: list):
    # Transformer chaque fichier en texte
    documents_as_text = {}
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            filepath = os.path.join(root, file)
            documents_as_text[filepath] = self.file_to_text(filepath)
            # file_to_text : Une fonction pour convertir un fichier en texte

    # Generer les mots clés pour chaque thème
    theme_keywords = {}
    for theme in themes:
        theme_keywords[theme] = generate_related_keywords(theme)

    # Scoring des documents par rapport à chaque thème
    document_scores = {}
    for filepath, file_text in documents_as_text.items():
        scores = {}
        for theme, keywords in theme_keywords.items():
            scores[theme] = self.calculate_score(file_text, keywords)
        document_scores[filepath] = scores
        # calculate_score : Une fonction qui évalue la similarité
        # entre le texte du document et les mots-clés associés à chaque thème

    # Organiser les fichiers basés sur le score le plus élevé et un seuil d'attribution
    seuil = 0.5
    for filepath, scores in document_scores.items():
        max_score_theme = max(scores, key=scores.get)
        if scores[max_score_theme] < seuil:
            # Déplacez le fichier dans le dossier "autre"
            self.move_file_to_folder(filepath, "autre")
        else:
            # Déplacez le fichier dans le dossier du thème correspondant
            self.move_file_to_folder(filepath, max_score_theme)
```

Testing :

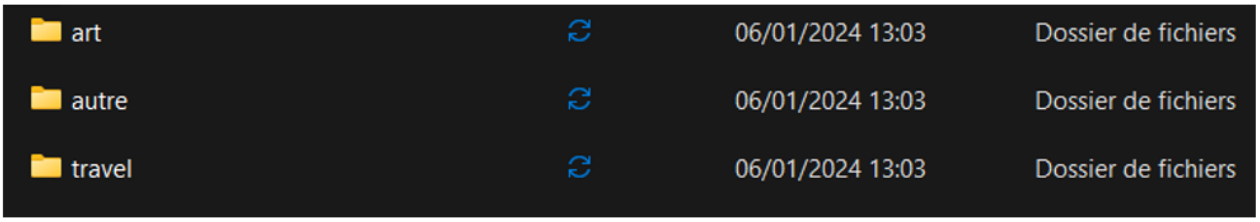
Organisation par thème :



Avant

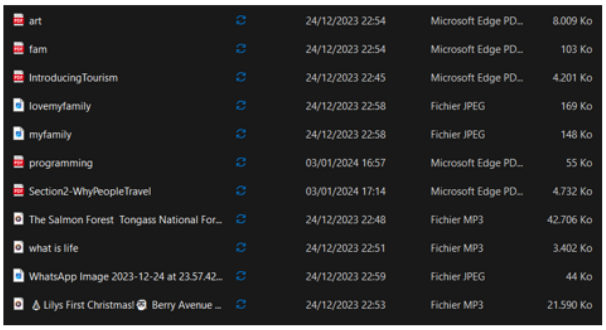


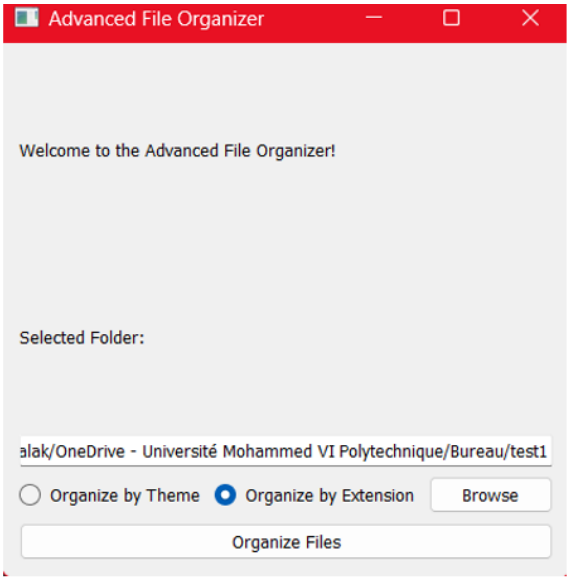
Après



Organisation par extensions :

Avant





Après