Rapport

Système d'Indexation et de Recherche d'Images par le Contenu

Encadré par Pr : M'hamed AIT KBIR

Fait par : **EL MEZIANE Chaïma**

OHMAD Abdessamade

Objectif

Le but de ce projet était de développer une application web permettant d'implémenter les fonctionnalités de base d'un système d'indexation et de recherche d'images par le contenu (CBIR). Cela inclut :

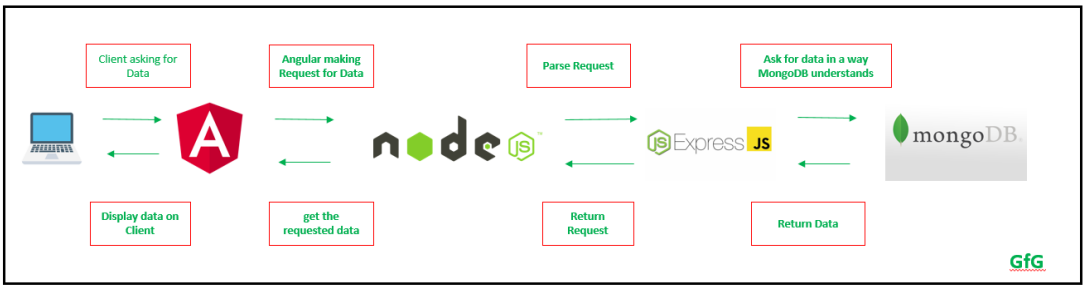
- Le chargement, le téléchargement et la suppression d'images.
- L'organisation d'images par catégories.
- La création de nouvelles images à partir de transformations (crop, redimensionnement, etc.).
- L'utilisation de services d'une API REST pour calculer des descripteurs d'images.
- La recherche d'images similaires avec deux approches : simple et avec retour de pertinence.

Architecture du Système

L'architecture du système repose sur une conception modulaire et intégrée combinant des technologies modernes pour assurer une expérience utilisateur fluide et des performances optimales. Elle est structurée comme suit :

- Backend : Flask (microframework Python) avec l'extension Flask-RESTful pour la gestion des API.

- Frontend : MEAN Stack (MongoDB, Express, Angular, Node.js) pour l'interface utilisateur, incluant l'authentification et la gestion des images.



- Swagger : Documentation et test des API.

Données

Le dataset utilisé dans ce projet est le RSSCN7 (Robust Scene Text Image Dataset), qui est conçu pour les tâches de reconnaissance de texte dans des images de scènes réelles. Ce dataset contient un total de 2800 images, réparties en 7 catégories distinctes. Chaque catégorie comprend 400 images, ce qui permet d'avoir une diversité de scènes et de contextes pour les tests et les expériences.

Voici les 7 catégories présentes dans le dataset RSSCN7 :

- **aGrass** : Cette catégorie regroupe des images de scènes naturelles, où l'herbe est présente dans l'environnement. Les images peuvent contenir du texte sur des surfaces herbeuses ou dans des paysages verdoyants.
- **bField** : Les images dans cette catégorie représentent des champs ou des étendues de terrain cultivé. Le texte peut apparaître sur des panneaux dans des environnements agricoles ou des scènes rurales.
- **cIndustry** : Cette catégorie contient des images de scènes industrielles, incluant des usines, des entrepôts, des panneaux de signalisation industriels, etc. Le texte est souvent vu dans un contexte de production ou d'activité industrielle.
- **dRiverLake** : Les images de cette catégorie représentent des paysages incluant des rivières ou des lacs. Le texte peut être visible sur des enseignes dans des zones proches de l'eau ou dans des environnements naturels.
- **eForest** : Cette catégorie contient des images de forêts ou de zones boisées. Le texte peut apparaître sur des panneaux d'information, des signalétiques ou même sur des éléments naturels du paysage forestier.
- **fResident** : Les images dans cette catégorie représentent des environnements résidentiels, tels que des maisons, des appartements ou des quartiers résidentiels. Les panneaux de rue, les enseignes et les écriteaux sont les types de texte fréquemment observés.
- **gParking** : Cette catégorie inclut des images d'aires de stationnement, de parkings en plein air ou sous-terrain. Le texte visible dans ces images peut être des panneaux de signalisation, des informations sur les places de stationnement ou des indications.

Query-Point Movement for CBIR

La méthode du Query-Point Movement (QPM) a pour objectif de réajuster un point de requête dans l'espace de caractéristiques de manière à le rapprocher des résultats pertinents et à l'éloigner des résultats non pertinents. En d'autres termes, le but du QPM est d'améliorer l'estimation du "point de requête idéal" en le déplaçant vers des exemples positifs et loin des exemples négatifs. Cette approche se base sur un mécanisme itératif où l'utilisateur fournit des retours pour affiner la recherche.

L'approche la plus connue du QPM a été développée par Rocchio dans le cadre de la recherche d'informations textuelles, et elle s'applique aussi dans le domaine de la recherche d'images par contenu (CBIR). La formule de mise à jour du point de requête est la suivante :

$$Q' = \alpha Q + \beta \left(\frac{1}{|D_R|} \sum_{i \in D_R} D_i \right) - \gamma \left(\frac{1}{|D_N|} \sum_{i \in D_N} D_i \right)$$

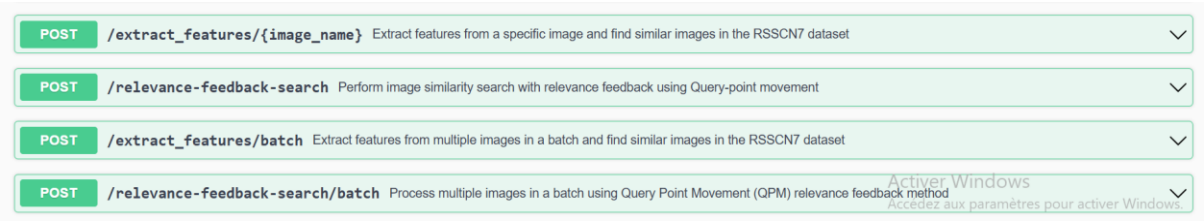
Où :

- Q et Q’ sont respectivement le point de requête original et le point de requête mis à jour.
- DR et DN représentent les ensembles des documents pertinents et non pertinents retournés par l'utilisateur.
- Di désigne les vecteurs de caractéristiques des images dans les ensembles pertinents et non pertinents.
- α, β, et γ sont des coefficients qui contrôlent l'impact relatif des différentes parties de la mise à jour du point de requête.

Dans cette formule, le point de requête est ajusté en fonction des retours de l'utilisateur : il est déplacé vers les images pertinentes (avec un facteur β) et loin des images non pertinentes (avec un facteur γ). L'objectif est ainsi d'affiner la recherche afin de rendre la requête plus précise pour les besoins de l'utilisateur.

Fonctionnalités Implémentées

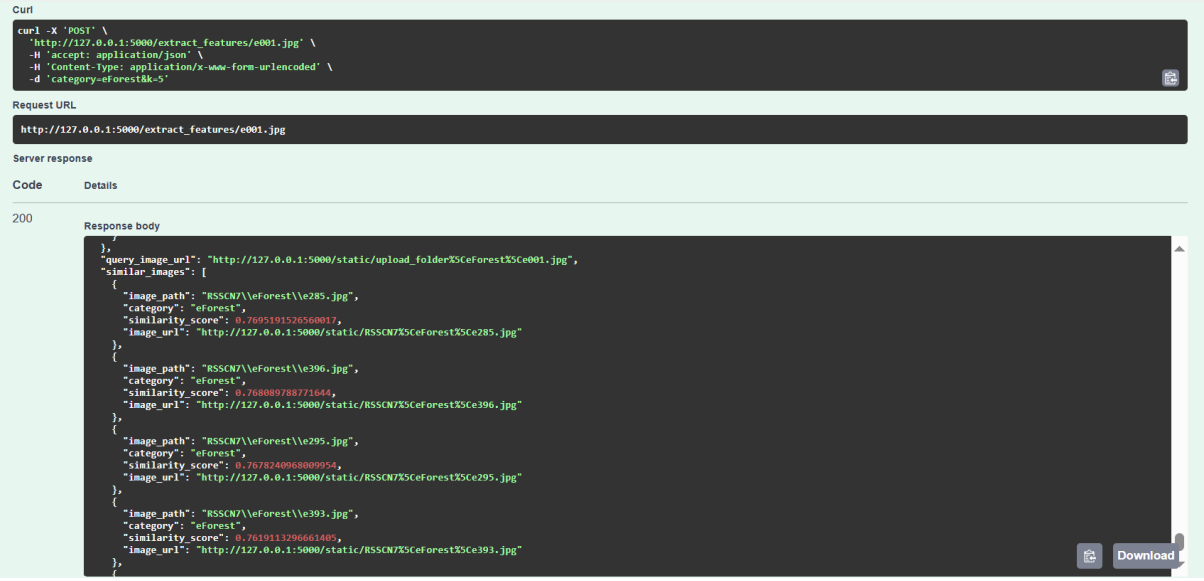
3.1 Backend Flask



3.1.1 Classe ExtractFeaturesResource

La classe ExtractFeaturesResource implémente une ressource de l'API REST pour extraire des caractéristiques d'une image spécifique et trouver des images similaires à partir de caractéristiques mises en cache. Voici les étapes et techniques utilisées dans son implémentation :

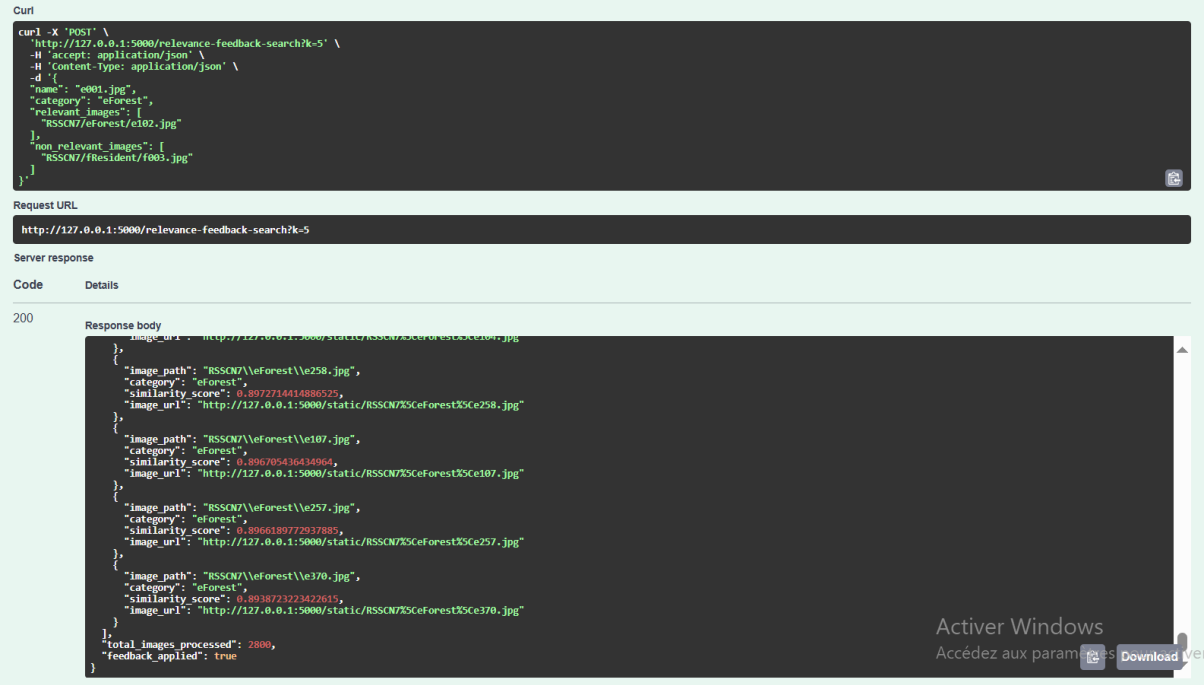
- **Extraction des paramètres de la requête :** Lors de la réception d'une requête POST, l'image et la catégorie sont extraites des données du formulaire via request.form. Le nombre d'images similaires à renvoyer (k) est également récupéré, avec une valeur par défaut de 10.
- **Validation des entrées :** La catégorie est vérifiée pour s'assurer qu'elle fait partie d'un ensemble prédéfini de catégories valides. Si la catégorie ou l'image est incorrecte, un message d'erreur est retourné avec un code HTTP 400 ou 404.
- **Extraction des caractéristiques de l'image** : Une fois l'image validée, les caractéristiques de l'image sont extraites à l'aide d'un extracteur de caractéristiques personnalisé, feature_extractor.extract_all_features.
- **Recherche dans les caractéristiques mises en cache :** La classe parcourt un répertoire de cache contenant les caractéristiques d'images issues de différentes catégories (RSSCN7 dataset). Les fichiers de caractéristiques sont chargés à partir de fichiers .pkl et comparés avec les caractéristiques de l'image de la requête.
- **Calcul de la similarité :** Pour chaque image dans le cache, la similarité est calculée en utilisant la méthode _calculate_similarity. Celle-ci prend en compte plusieurs types de caractéristiques (histogrammes de couleurs, textures, etc.) et utilise des métriques de comparaison adaptées, comme la distance du chi carré pour les histogrammes et la similarité cosinus pour les vecteurs de caractéristiques.
- **Tri des résultats :** Les images similaires sont triées en fonction de leur score de similarité, et un sous-ensemble des k images les plus similaires est renvoyé dans la réponse JSON.



3.1.2 Classe RelevanceFeedbackSearchResource

La classe RelevanceFeedbackSearchResource implémente une méthode de recherche d'images en utilisant la méthode de feedback de pertinence basée sur le Query Point Movement (QPM). Voici les étapes clés de son implémentation :

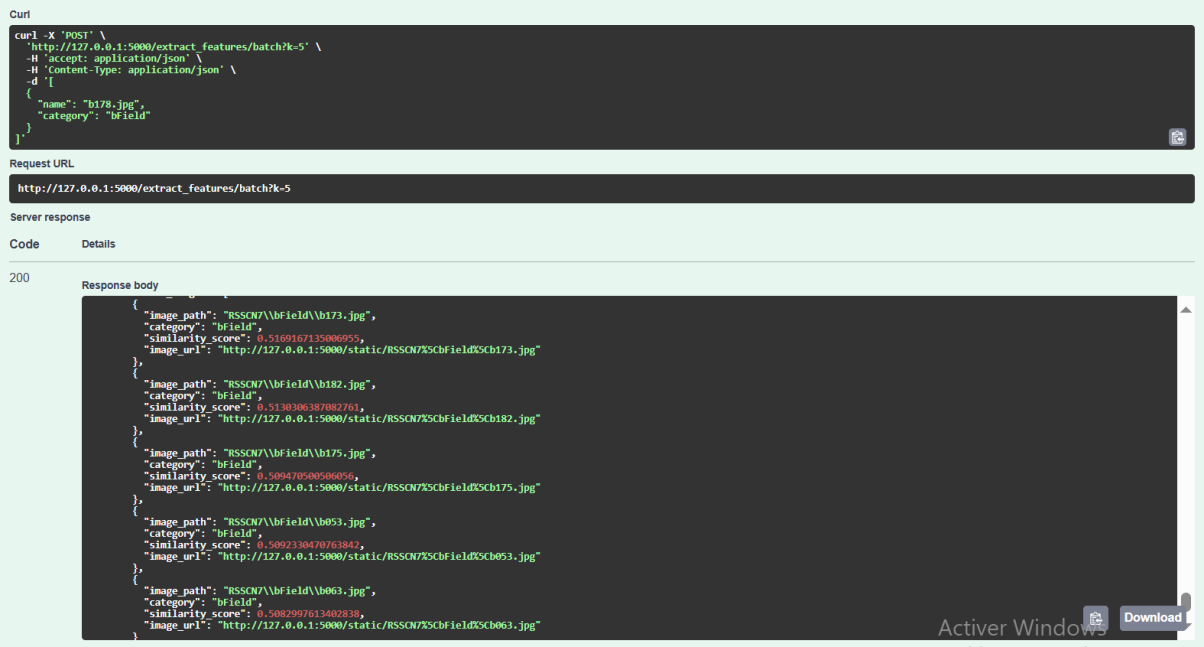
- **Validation des données de requête :** Lors de la réception d'une requête HTTP POST, la méthode vérifie d'abord si la requête contient des données JSON valides. Ensuite, elle s'assure que les données de l'image sont présentes et qu'elles incluent des champs essentiels comme name et category.
- **Gestion des paramètres et validation :** Des paramètres comme k, alpha, beta, et gamma sont extraits de la requête avec des valeurs par défaut. De plus, la catégorie de l'image est validée pour s'assurer qu'elle fait partie des catégories autorisées (par exemple, 'aGrass', 'bField', etc.).
- **Validation des chemins d'images :** La méthode _validate_image_paths est utilisée pour valider les chemins des images pertinentes et non pertinentes, en vérifiant leur existence sur le serveur.
- **Extraction des caractéristiques de l'image :** Une fois l'image cible trouvée dans le dossier approprié, les caractéristiques de l'image sont extraites à l'aide de la fonction feature_extractor.extract_all_features.
- **Application du QPM :** Si des images pertinentes et non pertinentes sont fournies, la méthode _apply_qpm ajuste les caractéristiques de l'image cible à l'aide de la formule de Rocchio pour les histogrammes et les vecteurs de caractéristiques comme les Gabor et HOG.
- **Calcul de la similarité :** Pour chaque image dans le cache des caractéristiques, la similarité avec l'image cible est calculée en utilisant des mesures spécifiques, comme la distance du chi carré pour les histogrammes de couleur et des calculs de similarité pour d'autres types de caractéristiques (couleurs dominantes, HOG, etc.).
- **Renvoi des résultats :** Enfin, les résultats sont triés par score de similarité et renvoyés au format JSON. Cela inclut des informations sur l'image requêtée, les images similaires, et le nombre total d'images traitées.



3.1.3 Classe BatchFeatureExtractionResource

La classe BatchFeatureExtractionResource permet d'extraire des caractéristiques de plusieurs images en lot en utilisant les caractéristiques mises en cache pour effectuer des comparaisons. Voici les principales étapes et techniques utilisées dans son implémentation :

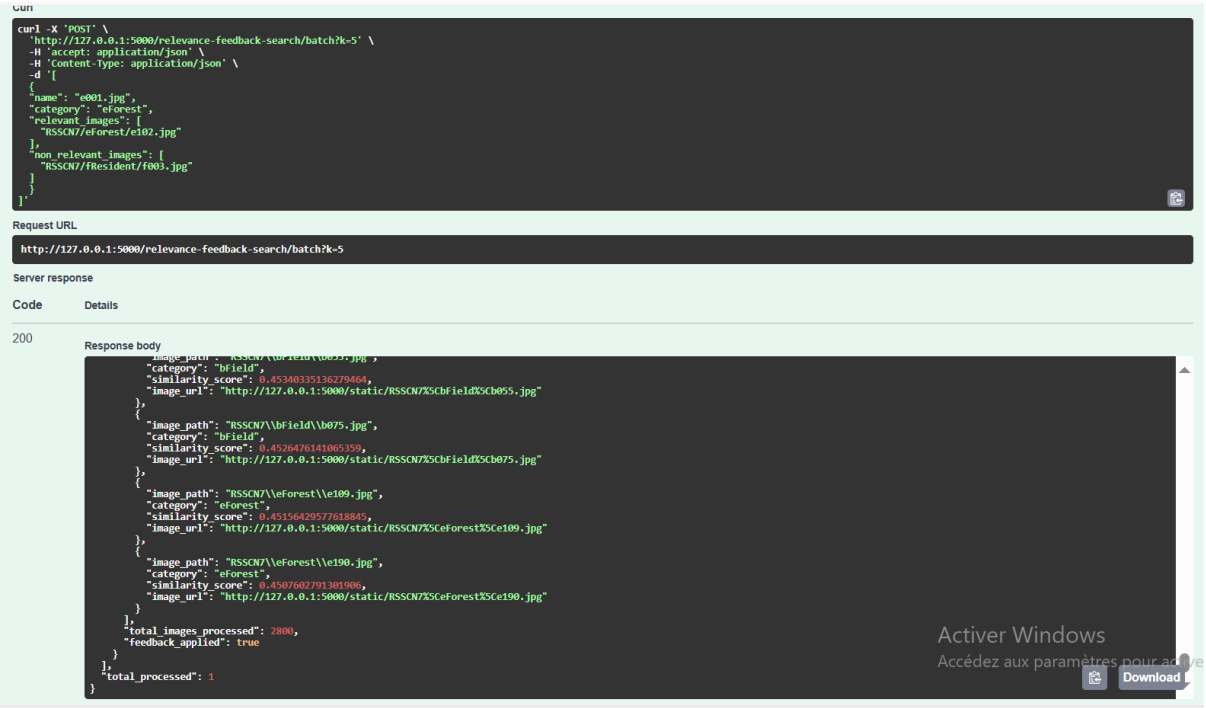
- **Validation des données de la requête :** Avant tout traitement, la classe vérifie si la requête contient des données JSON valides et si elle inclut des images sous forme de liste ou de dictionnaire. Si aucune image n'est fournie ou si les données sont mal formatées, une réponse d'erreur est renvoyée.
- **Traitement par lot :** Pour chaque image du lot, le nom et la catégorie sont vérifiés. Si des informations manquent ou si la catégorie est invalide, un message d'erreur est ajouté au résultat du lot.
- **Extraction des caractéristiques :** Pour chaque image valide, la méthode feature_extractor.extract_all_features() est utilisée pour extraire les caractéristiques de l'image. Ces caractéristiques sont ensuite comparées avec celles des images mises en cache dans le répertoire feature_cache_base.
- **Calcul de la similarité :** La fonction _calculate_similarity() calcule un score de similarité entre les caractéristiques de l'image requête et celles des images du cache, en utilisant plusieurs métriques de comparaison (histogrammes de couleurs, couleurs dominantes, Gabor, LBP, etc.). Chaque type de caractéristique est pondéré pour donner plus d'importance aux éléments jugés les plus pertinents.
- **Utilisation du cache :** Les caractéristiques des images sont stockées sous forme de fichiers .pkl dans un répertoire de cache. Lors de la comparaison, ces fichiers sont chargés, et les scores de similarité sont calculés entre les caractéristiques extraites de l'image requête et celles des images mises en cache.
- **Résultats du lot :** Après le traitement de chaque image, les résultats sont triés par score de similarité et renvoyés sous forme de JSON avec les images similaires et les caractéristiques extraites. Si une erreur survient à n'importe quelle étape, elle est capturée et ajoutée aux résultats.



3.1.4 Classe BatchRelevanceFeedbackSearchResource

La classe BatchRelevanceFeedbackSearchResource a été implémentée en utilisant une méthode de recherche d'images en utilisant la méthode de feedback de pertinence basée sur le Query Point Movement (QPM) sur un lot d'images. Voici les principales étapes et techniques utilisées dans l'implémentation :

- **Gestion de la requête JSON :** La classe commence par vérifier que la requête inclut des données JSON. Si ce n'est pas le cas, une erreur est renvoyée. Les données peuvent être envoyées sous forme de liste ou de dictionnaire. Si aucune image n'est fournie, un message d'erreur est renvoyé.
- **Validation des catégories :** Les catégories valides sont spécifiées, telles que aGrass, bField, etc. Chaque image doit appartenir à l'une de ces catégories pour être traitée. Si la catégorie est invalide, un message d'erreur est renvoyé.
- **Extraction des caractéristiques :** Pour chaque image dans le lot, un chemin d'accès est construit et vérifié. Si l'image n'existe pas, une erreur est générée. Sinon, les caractéristiques de l'image sont extraites en utilisant un extracteur de caractéristiques (fonction extract_all_features).
- **Application du feedback de pertinence avec QPM :** Lors de la recherche par similarité, les résultats sont affinés en appliquant la méthode Query Point Movement (QPM), une technique qui permet de modifier les caractéristiques d'une requête en fonction des images jugées pertinentes ou non pertinentes. Si des images pertinentes et non pertinentes sont fournies, la méthode _apply_qpm est utilisée pour ajuster les caractéristiques originales de la requête, en mettant à jour les descripteurs de caractéristiques (histogrammes de couleurs, vecteurs de Gabor, HOG, etc.). Cette mise à jour est réalisée en appliquant la formule de Rocchio, où les centroids des images pertinentes et non pertinentes sont calculés et utilisés pour ajuster les caractéristiques de la requête.
- **Tri des résultats par score de similarité** : Les résultats de la recherche sont triés par score de similarité décroissant et renvoyés avec les informations pertinentes sur chaque image, comme l'URL et le score de similarité.



3.2 Frontend MEAN

3.2.1 Authentification des utilisateurs

L'authentification des utilisateurs a été implémentée dans le frontend de l'application en utilisant la stack MEAN (MongoDB, Express.js, Angular, Node.js). Voici les étapes principales :

- **Enregistrement des utilisateurs :**

Les utilisateurs peuvent créer un compte via un formulaire sécurisé. Les données sont transmises au backend Node.js, où les mots de passe sont hachés avant d'être stockés dans MongoDB.

- **Connexion et génération de JWT :**

Lorsqu'un utilisateur se connecte, ses identifiants sont vérifiés. En cas de succès, un JSON Web Token (JWT) est généré. Ce token inclut les informations nécessaires pour identifier l'utilisateur (comme son ID) et a une durée de validité limitée pour des raisons de sécurité.

- **Stockage sécurisé du token :**

Le token JWT est envoyé au frontend Angular et stocké dans le localStorage ou les cookies (selon le niveau de sécurité souhaité). Cela permet de maintenir la session de l'utilisateur.

- **Validation des requêtes via le middleware :**

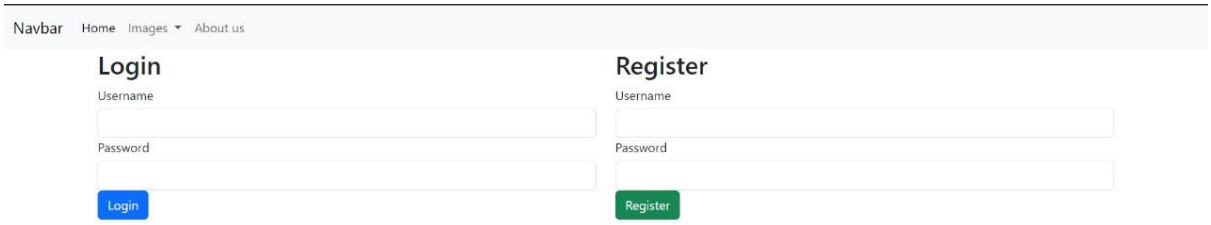
Chaque requête nécessitant une authentification (comme la gestion des images ou l'accès aux résultats de recherche) passe par un middleware sur le backend Express.js. Ce middleware valide le JWT et permet uniquement aux utilisateurs authentifiés d'accéder aux services.

- **Protection des routes Angular :**

Sur le frontend, un guard Angular empêche l'accès aux pages protégées si l'utilisateur n'est pas connecté ou si le JWT est expiré.

- **Déconnexion et gestion des sessions :**

Les utilisateurs peuvent se déconnecter, ce qui supprime le JWT côté client. Les sessions expirées sont automatiquement invalidées grâce à la durée limitée du token.



3.2.1 Gestion des images

La gestion des images dans le système repose sur les opérations CRUD (Create, Read, Update, Delete), avec une interface utilisateur conviviale développée sous Angular et des fonctionnalités backend robustes utilisant Node.js et MongoDB. Voici les principales fonctionnalités implémentées :

- **Upload et gestion des images :**
  - L'utilisateur peut charger une ou plusieurs images via un formulaire d'upload sur l'interface Angular.
  - Les images sont transférées au backend via une API REST Express.js et stockées dans un système de fichiers ou une base de données adaptée (comme MongoDB avec GridFS pour les fichiers volumineux).
  - Chaque image est associée à des métadonnées (nom, taille, type, catégorie, etc.) qui sont enregistrées dans la base MongoDB pour permettre une gestion et une recherche efficaces.

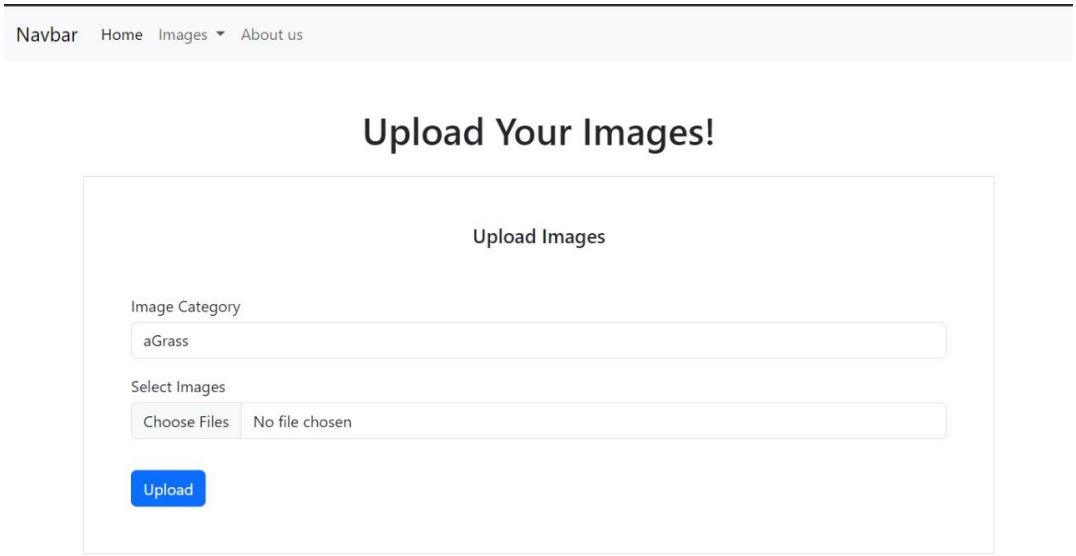


Image Category

aGrass

Select Images

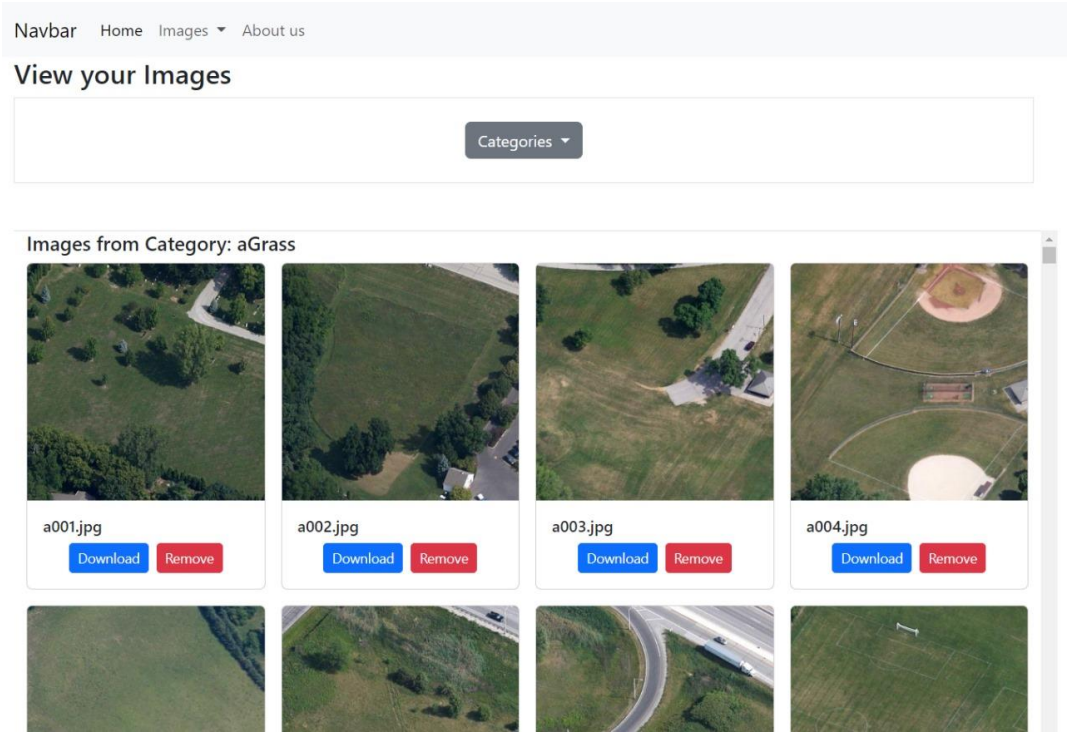
Choose Files | green-grass-above-11302574-2888090284.jpg

Preview Image

Upload

Transform Image

- **Recherche par nom d'image, et catégorie :**
  - Une fonctionnalité de recherche permet de filtrer les images par leur nom. L'utilisateur entre le nom (ou une partie du nom) dans un champ de recherche sur l'interface Angular.
  - Le backend effectue une recherche textuelle dans MongoDB pour récupérer les images correspondantes, en renvoyant une liste paginée des résultats.
  - Les images trouvées sont affichées sur l'interface avec leurs métadonnées, permettant à l'utilisateur de les visualiser ou d'agir dessus.



- **Téléchargement et suppression des images :**
  - Téléchargement : Chaque image listée peut être téléchargée directement en cliquant sur un bouton de téléchargement. Le backend Express.js renvoie le fichier via une réponse HTTP pour que l'utilisateur puisse l'enregistrer localement.
  - Suppression : Un utilisateur peut supprimer une image via un bouton dédié. Une requête DELETE est envoyée au backend, qui supprime l'image du système de fichiers et efface ses métadonnées de MongoDB.
  - Des mécanismes de confirmation sont intégrés dans l'interface Angular pour éviter toute suppression accidentelle.

Ces fonctionnalités offrent une gestion complète et intuitive des images, essentielle pour la manipulation et l'organisation des données nécessaires au système de recherche par contenu.

3.2.2 Recherche des images

La fonctionnalité de recherche des images est un élément clé du système, permettant de retrouver des images similaires en se basant sur leur contenu visuel. Elle est divisée en deux parties : la recherche simple et la recherche avec retour de pertinence, et chacune peut être effectuée sur une image unique ou sur un lot d'images.

Recherche simple

La recherche simple consiste à trouver les images les plus similaires à une image requête en fonction de leurs descripteurs visuels.

- Processus :
  - L'utilisateur sélectionne une image requête ou un ensemble d'images depuis l'interface Angular.
  - Une requête POST est envoyée au backend avec l'image ou les images sélectionnées.

- Les descripteurs visuels (histogrammes de couleurs, textures de Gabor, moments de Hu, etc.) sont calculés pour l'image requête à l'aide des services REST fournis par Flask.
- Une comparaison est effectuée avec les descripteurs des images stockées dans la base pour calculer les distances de similarité (par exemple, la distance euclidienne).
- Les images les plus proches sont renvoyées et affichées sur l'interface utilisateur sous forme de liste ou de galerie.

Recherche avec retour de pertinence

La recherche avec retour de pertinence améliore les résultats en tenant compte des retours de l'utilisateur sur les images pertinentes ou non pertinentes. Cette méthode utilise l'algorithme Query Point Movement (QPM) pour ajuster dynamiquement les descripteurs de l'image requête.

- Processus :
  - L'utilisateur sélectionne une image requête ou un lot d'images comme dans la recherche simple.
  - Après avoir visualisé les résultats initiaux, l'utilisateur marque certaines images comme pertinentes ou non pertinentes.
  - Ces retours sont envoyés au backend, où l'algorithme QPM ajuste les descripteurs de l'image requête en déplaçant son point dans l'espace des caractéristiques (en fonction des poids alpha, beta et gamma).
  - Une nouvelle recherche est effectuée avec les descripteurs mis à jour pour affiner les résultats.
  - Les résultats révisés sont renvoyés et affichés, offrant une meilleure pertinence par rapport aux besoins de l'utilisateur.

3.2.3 Transformation des images

La fonctionnalité de transformation des images permet aux utilisateurs de créer de nouvelles images en appliquant différentes transformations sur les images existantes. Ces opérations sont utiles pour enrichir la base d'images, générer des variantes des images originales, et explorer les impacts de ces transformations sur les résultats de recherche par contenu. Les transformations implémentées incluent le recadrage (crop) et le changement d'échelle (resize).

Recadrage (Crop)

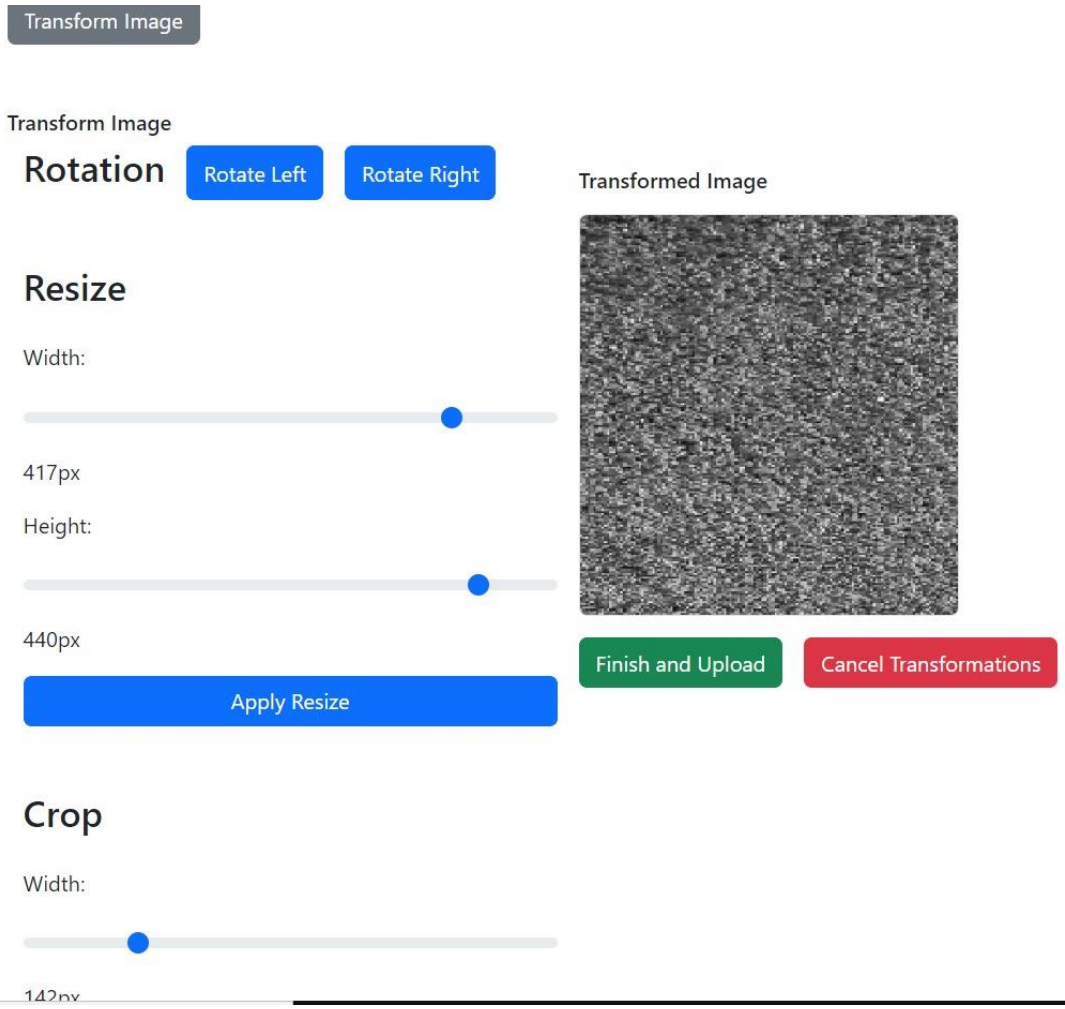
Le recadrage permet de sélectionner une région spécifique de l'image et de générer une nouvelle image basée sur cette région.

- Processus :
  - L'utilisateur charge une image et définit une zone de recadrage via l'interface Angular (par exemple, en spécifiant les coordonnées ou en traçant un rectangle).
  - Une requête POST contenant les paramètres de recadrage est envoyée au backend.
  - L'image est traitée sur le serveur pour extraire la région spécifiée.
  - La nouvelle image recadrée est enregistrée dans la base et renvoyée à l'utilisateur pour prévisualisation.

Changement d'échelle (Resize)

Le changement d'échelle permet de modifier les dimensions d'une image tout en conservant son contenu visuel global.

- Processus :
    - L'utilisateur sélectionne une image et fournit les nouvelles dimensions (largeur et hauteur) via l'interface utilisateur.
    - Une requête POST contenant les dimensions spécifiées est envoyée au backend.
    - Le serveur utilise une interpolation (bilinéaire ou bicubique) pour redimensionner l'image.
    - L'image redimensionnée est sauvegardée dans la base et mise à disposition pour téléchargement ou utilisation dans les recherches.



3.3 Documentation et Tests

- **Swagger :**

Fournit une documentation interactive des endpoints de l'API.

Permet de tester les routes (e.g., upload, transformation, recherche).

- **Tests des API :**

Validation des résultats des transformations et des similarités retournée

Conclusion

Ce projet a permis de concevoir et de développer un système complet d'indexation et de recherche d'images par le contenu, intégrant des fonctionnalités avancées telles que la recherche simple, la recherche avec retour de pertinence, et la transformation des images. En combinant des technologies modernes comme Flask pour le backend et la stack MEAN pour le frontend, nous avons pu répondre aux besoins fonctionnels tout en assurant une architecture performante, modulaire, et évolutive.

Le projet s'est appuyé sur des concepts fondamentaux du traitement d'images, tels que les descripteurs de contenu (histogrammes de couleurs, moments de Hu, etc.), et leur implémentation a démontré l'efficacité des méthodes pour extraire et comparer les caractéristiques visuelles. L'intégration des services RESTful et des techniques de retour de pertinence a enrichi l'expérience utilisateur en rendant le système plus précis et interactif.