LuminaShare

Arnaud Gomes, Kamiel De Vos Le plus grand, Soraya Benachour

1 Introduction

Ce document présente les besoins nécessaires au développement d'une application de recherche d'image par similarité avec une architecture de type client-serveur s'appuyant sur une base de données pour l'indexation des images. Afin de permettre aux groupes d'approfondir un sujet qui les intéresse plus particulièrement, on propose la structure suivante.

Noyau commun : Chaque groupe devra réaliser l'implémentation du fonctionnement central de l'application client-serveur. Elle est décrite dans la section 2. Notez qu'une partie des besoins a été développée dans les TP communs. Le code du noyau commun fera l'objet d'un rendu intermédiaire.

Extensions : Des suggestions d'extensions seront proposées en cours (amélioration de l'interface utilisateur, descripteurs d'image plus avancés, pagination des résultats, sessions utilisateur persistantes, gestion d'une arborescence de fichiers image, etc), mais vous êtes invités à faire vos propres propositions.

Chaque groupe peut faire évoluer ce document avec l'aval de son chargé de TD. Le cahier des besoins fera partie des rendus.

L'application devra permettre de traiter les images couleur enregistrées aux formats suivants :
— `JPEG`
— `PNG`

2 Noyau commun

2.1 Serveur

Besoin 1 : Initialiser un ensemble d'images présentes sur le serveur

Description : Lorsque le serveur est lancé, il doit charger toutes les images présentes à l'intérieur du dossier `images` et les indexer dans une base de données (cf. Besoin 3). Ce dossier `images` doit exister à l'endroit où est lancé le serveur. Seuls les fichiers images correspondants aux formats d'image reconnus doivent être traités. Les sous-dossiers du dossier `images` ne seront pas traités.

Gestion d'erreurs : Si le dossier `images` n'existe pas depuis l'endroit où a été lancé le serveur, une erreur explicite doit être levée.

Tests :

1. Lancement de l'exécutable depuis un environnement vide, une erreur doit se déclencher indiquant que le dossier `images` n'est pas présent.
2. Mise en place d'un dossier de test contenant des images aux formats reconnus ainsi que des documents avec des extensions non-reconnues comme étant des images (e.g., `.txt`).

Besoin 2 : Gérer les images présentes sur le serveur

Description : Le serveur gère un ensemble d'images. Il permet d'accéder aux données brutes de chaque image ainsi qu'aux méta-données nécessaires aux réponses aux requêtes (identifiant, nom de fichier, taille de l'image, format,...). Le serveur peut :

1. accéder à une image via son identifiant,
2. supprimer une image via son identifiant,
3. ajouter une image,
4. construire la liste des images disponibles (composée uniquement des métadonnées).

Le serveur garantit la persistance des données : le dossier `images` contient à tout moment l'ensemble des images disponibles qui sont également indexées dans la base de données.

Besoin 3 : Indexer une image

Description : Le serveur permet d'indexer une image c'est-à-dire d'ajouter dans une base de données un enregistrement correspondant à l'image (identifiant et descripteur). Les descripteurs supportés sont l'histogramme 2D Teinte/Saturation et l'histogramme 3D RGB de l'image.

Besoin 4 : Rechercher des images similaires à une image donnée

Description : Le serveur permet de construire la liste des `N` images les plus similaires à une image donnée pour un descripteur donné.

2.2 Communication

Pour l'ensemble des besoins, des codes d'erreurs à renvoyer sont précisés dans le paragraphe "Gestion d'erreurs" (à compléter si nécessaire).

Besoin 5 : Transférer la liste des images existantes

Description : La liste des images présentes sur le serveur doit être envoyée par le serveur lorsqu'il reçoit une requête `GET` à l'adresse `/images`.
Le résultat sera fourni au format `JSON`, sous la forme d'un tableau contenant pour chaque image un objet avec les informations suivantes :

Id : L'identifiant auquel est accessible l'image (type `long`)

Name : Le nom du fichier qui a servi à construire l'image (type `string`)

Type : Le type de l'image (type `org.springframework.http.MediaType`)

Size : Une description de la taille de l'image, par exemple `640*480` pour une image de 640×480 pixels (type `string`)

Tests : Pour le dossier de tests spécifié dans Besoin 1, la réponse attendue doit être comparée à la réponse reçue lors de l'exécution de la commande.

Besoin 6 : Ajouter une image

Description : L'envoi d'une requête `POST` à l'adresse `/images` avec des données de type multi-media dans le corps de la requête doit ajouter une image à celles disponibles sur le serveur (voir Besoin 2).

Gestion d'erreurs :

201 Created : La requête s'est bien exécutée et l'image est à présent sur le serveur.

415 Unsupported Media Type : La requête a été refusée car le serveur ne supporte pas le format reçu (par exemple `EXR`).

Besoin 7 : Récupérer une image

Description : L'envoi d'une requête `GET` à une adresse de la forme `/images/id` doit renvoyer l'image stockée sur le serveur avec l'identifiant `id` (entier positif). En cas de succès, l'image est retournée dans le corps de la réponse.

Gestion d'erreurs :

200 OK : L'image a bien été récupérée.

404 Not Found : Aucune image existante avec l'identifiant `id`.

Besoin 8 : Supprimer une image

Description : L'envoi d'une requête `DELETE` à une adresse de la forme `/images/id` doit effacer l'image stockée avec l'identifiant `id` (entier positif). Voir Besoin 2.

Gestion d'erreurs :

200 OK : L'image a bien été effacée.

404 Not Found : Aucune image existante avec l'identifiant `id`.

Besoin 9 : Transférer la liste des images les plus similaires à une image donnée

Description : Lors d'une requête `GET` à une adresse de la forme `/images/id/similar?number=N&descriptor=DESCR` le serveur envoie la liste des `N` images les plus similaires à l'image d'identifiant `id` selon le descripteur `DESCR`.
Le résultat sera fourni au format `JSON`, sous la forme d'un tableau contenant les informations de chaque image (comme pour le Besoin 5)

Gestion d'erreurs :

200 OK : La requête a bien été traitée.

400 Bad Request : La requête ne peut être traitée, par exemple si la valeur du paramètre `descriptor` ne correspond pas à un descripteur disponible.

404 Not Found : Aucune image existante avec l'indice `id`.

2.3 Client

Les actions que peut effectuer l'utilisateur côté client induisent des requêtes envoyées au serveur. En cas d'échec d'une requête, le client doit afficher un message d'erreur explicatif.

Besoin 10 : Parcourir les images disponibles sur le serveur

Description : L'utilisateur peut visualiser les images disponibles sur le serveur. La présentation visuelle peut prendre la forme d'un carroussel ou d'une galerie d'images. On suggère que chaque vignette contenant une image soit de taille fixe (relativement à la page affichée). Suivant la taille de l'image initiale la vignette sera complètement remplie en hauteur ou en largeur.

Besoin 11 : Sélectionner une image et afficher les images similaires

Description : L'utilisateur peut cliquer sur la vignette correspondant à une image. L'image est affichée sur la page. L'utilisateur peut visualiser les méta-données de l'image. L'utilisateur peut choisir d'afficher les images similaires disponibles. Il peut préciser le nombre d'images similaires à afficher et le descripteur utilisé pour la mesure de similarité. Pour chaque image similaire est affiché un score de similarité.

Besoin 12 : Enregistrer une image sur disque

Description : L'utilisateur peut sauvegarder dans son système de fichier une image chargée.

Besoin 13 : Ajouter une image aux images disponibles sur le serveur

Description : L'utilisateur peut ajouter une image choisie dans son système de fichier aux images disponibles sur le serveur. Cet ajout est persistant (un fichier est ajouté côté serveur).

Besoin 14 : Supprimer une image

Description : Le client peut choisir de supprimer une image préalablement sélectionnée. Elle n'apparaîtra plus dans les images disponibles sur le serveur. Cette suppression est persistante (un fichier est supprimé côté serveur).

2.4 Besoins non-fonctionnels

Besoin 15 : Intégration continue

Description : Les outils de Gitlab permettant le développement partagé et l'intégration continue seront utilisés pour :

- le versionnage,
- la gestion des tickets,
- la compilation de l'application,
- l'exécution des tests.

Besoin 16 : Compatibilité du serveur

Description : La partie serveur de l'application sera écrite en Java (JDK 17) avec les bibliothèques suivantes :

- `org.springframework.boot` : version 3.2.x
- `org.boofcv` : version 1.1.2
- `org.postgresql` : version 42.7.1
- `com.pgvector` : version 0.1.4

La base de données utilisée sera PostgreSQL avec l'extension `pgvector`. Spring JDBC sera utilisé pour faire le lien avec la base.

Le fonctionnement du serveur devra être éprouvé sur au moins un des environnements suivants :

- Windows $\geq$ 10
- Ubuntu $\geq$ 20.04
- Debian Bookworm
- MacOS $\geq$ 11

Besoin 17 : Compatibilité du client

Description : Le client sera écrit en TypeScript et s'appuiera sur la version `3.x` du framework `Vue.js`.

Le client devra être testé sur au moins l'un des navigateurs Web suivants, la version à utiliser n'étant pas imposée :

- Safari
- Google Chrome
- Firefox

Besoin 18 : Documentation d'installation et de test

Description : La racine du projet devra contenir un fichier `README.md` indiquant au moins les informations suivantes :

- Système(s) d'exploitation sur lesquels votre serveur a été testé, voir Besoin 16.
- Navigateur(s) web sur lesquels votre client a été testé incluant la version de celui-ci, voir Besoin 17.