LuminaShare

Arnaud Gomes, Kamiel De Vos Le plus, 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 vérifier l'existence du dossier `images` à l'endroit où le serveur est exécuté. S'il n'existe pas, le serveur doit créer ce dossier automatiquement. Ensuite, il charge toutes les images présentes dans ce dossier et les indexe dans une base de données (cf. Besoin 3). Seuls les fichiers images correspondants aux formats reconnus (JPEG, PNG) doivent être traités. Les sous-dossiers du dossier `images` ne seront pas traités.

Gestion d'erreurs : Si la création du dossier `images` échoue (par exemple, en raison de permissions insuffisantes), une erreur explicite doit être levée avec un message indiquant la cause.

Tests :

1. Lancement du serveur dans un environnement où le dossier `images` n'existe pas : vérifier que le dossier est créé automatiquement et que le serveur démarre sans erreur.
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`) et vérifier que seules les images JPEG et PNG sont indexées..

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.

Besoin 5 : Système d'authentification utilisateur

Description : Le serveur doit implémenter un système d'authentification basique avec identifiants uniques et mots de passe. Il stocke les données utilisateur de façon sécurisée dans la base de données :

- userid (identifiant unique choisi par l'utilisateur, clé primaire)
- name (pseudo affiché)
- bio (courte description de l'utilisateur)
- password (mot de passe chiffré)

Les mots de passe sont stockés de manière sécurisée à l'aide d'un algorithme de hachage moderne (BCrypt).

Gestion d'erreurs : En cas d'échec d'authentification, le serveur renvoie un code d'erreur 401 (Unauthorized).

Besoin 6 : Gestion des utilisateurs et de leurs contenus

Description : Le serveur associe chaque image à son utilisateur créateur via une clé étrangère (userid). Les utilisateurs peuvent :

- Consulter leur profil et leurs images
- Visualiser les profils des autres utilisateurs
- Gérer uniquement leurs propres images

Les images disposent d'un attribut "public" (booléen) qui détermine leur visibilité.

Besoin 7 : Informations enrichies pour les images

Description : Chaque image stockée dans la base de données contient des informations supplémentaires :

- userDescription (texte libre décrivant l'image)
- tags (liste de mots-clés associés à l'image)
- userid (identifiant de l'auteur de l'image)
- public (booléen indiquant si l'image est accessible à tous)

Tests :

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 8 : 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 9 : Routes pour la gestion des utilisateurs

Description : Le serveur expose les routes suivantes pour la gestion des utilisateurs :
- `/login` (POST) : Authentification d'un utilisateur
- `/register` (POST) : Création d'un nouvel utilisateur
- `/user/userid` (GET) : Récupération des informations d'un utilisateur
- `/user/userid/images` (GET) : Récupération des images d'un utilisateur

Gestion d'erreurs :

200 OK : La requête a été traitée avec succès

201 Created : L'utilisateur a été créé avec succès

400 Bad Request : Format de données incorrect

401 Unauthorized : Authentification échouée

404 Not Found : Utilisateur non trouvé

Besoin 10 : Routes pour les images

Description : Le serveur expose les routes suivantes pour la gestion des images :
- `/images/imageId/edit` (PUT) : Modification des informations d'une image (userDescription, tags)
- `/images/search` (GET) : Recherche d'images par tags ou auteur (tags, name)

Gestion d'erreurs :

200 OK : La requête a été traitée avec succès

400 Bad Request : Format de données incorrect

401 Unauthorized : Authentification échouée

403 Forbidden : Accès non autorisé (image privée ou non propriétaire)

404 Not Found : Image non trouvée

Besoin 11 : Ajouter une image

Description : L'envoi d'une requête `POST` à l'adresse `/images` avec des données de type multimedia 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 12 : 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 13 : 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 14 : 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 8)

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 15 : 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 16 : Interface d'authentification

Description : L'utilisateur peut s'inscrire et se connecter via des formulaires dédiés. Une fois connecté, il peut accéder à ses fonctionnalités privées.

Besoin 17 : Page d'accueil style Pinterest

Description : La page d'accueil présente une galerie d'images dans un style inspiré de Pinterest, avec des vignettes de différentes tailles organisées en colonnes. Les images publiques de tous les utilisateurs y sont affichées.

Besoin 18 : Page de profil utilisateur

Description : Chaque utilisateur dispose d'une page de profil affichant :

- Son identifiant(userid), pseudo (name) et sa biographie (bio)
- Une galerie de ses images publiques
- Pour le propriétaire du profil : toutes ses images (publiques et privées)

Le propriétaire du profil peut également ajouter, supprimer ou éditer ses images depuis cette page.

Besoin 19 : Barre de recherche

Description : Une barre de recherche permet à l'utilisateur de filtrer les images par :

- Mots-clés (tags)
- Identifiant utilisateur (userid)

Besoin 20 : 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 21 : Enregistrer une image sur disque

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

Besoin 22 : 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 23 : 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).

Besoin 24 : Prévisualiser l'application d'un filtre sur une image

Description : L'utilisateur peut sélectionner un filtre à appliquer à une image sélectionnée. Dans une séction prédéfinis, une image avec le filtre appliqué apparaîtra en temps réeel

Besoin 25 : Sauvegarder une image filtrée dans la galerie personnelle

Description : L'utilisateur peut sauvegarder une image après application d'un filtre (par exemple, redimensionnement, rotation) directement dans sa galerie personnelle sur le serveur. La nouvelle image conserve les mêmes paramètres que l'image originale, tels que la visibilité (publique ou privée). Le nom du fichier doit être unique pour éviter les conflits.

Gestion d'erreurs : Si l'enregistrement échoue (par exemple, en raison d'un problème de connexion ou d'un nom de fichier déjà existant), un message d'erreur explicatif doit être affiché à l'utilisateur.

2.4 Traitement d'images

Besoin 26 : Redimensionnement des images

Description : L'utilisateur peut redimensionner une image sélectionnée à une taille standard (des valeurs standard pour une hauteur et largeuer selon le beosin ou le choix) pour optimiser son stockage ou son affichage. La fonctionne redimensionne bien l'image selon les valeur qu'on propose mais la qualité d'image se diminue.

Gestion d'erreurs : Si l'image ne peut pas être redimensionnée (par exemple, en cas de corruption), un message d'erreur explicatif doit être affiché à l'utilisateur.

Besoin 27 : Inversion des couleurs

Description : L'utilisateur peut inverser les couleurs d'une image sélectionnée pour obtenir un effet de négatif (par exemple, le blanc devient noir, le rouge devient cyan).

Gestion d'erreurs : Si l'image est invalide ou corrompue, un message d'erreur explicatif doit être affiché à l'utilisateur.

Besoin 28 : Miroir de l'image

Description : L'utilisateur peut créer une version miroir de l'image sélectionnée, soit horizontalement (inversion gauche-droite), soit verticalement (inversion haut-bas).

Gestion d'erreurs : Si l'image ne peut pas être traitée (par exemple, en cas de corruption), un message d'erreur explicatif doit être affiché à l'utilisateur.

Besoin 29 : Rotation des images

Description : L'utilisateur peut faire pivoter une image sélectionnée selon un angle spécifié (par exemple, 90°, 180°, ou 270°) pour corriger son orientation ou appliquer un effet visuel.

Gestion d'erreurs : Si l'image ne peut pas être pivotée (par exemple, en cas de corruption ou d'angle invalide), un message d'erreur explicatif doit être affiché à l'utilisateur.

Besoin 30 : Réglage de la luminosité

Description : L'utilisateur peut augmenter ou diminuer la luminosité de l'image sélectionnée.

Besoin 31 : Filtres de flou

Description : L'utilisateur peut appliquer un flou à l'image sélectionnée. Il peut définir le filtre appliqué (moyen ou gaussien) et choisir le niveau de flou. La convolution est appliquée sur les trois canaux R, G et B.

2.5 Besoins non-fonctionnels

Besoin 32 : Conteneurisation avec Docker

Description : L'application sera conteneurisée à l'aide de Docker pour simplifier le déploiement.

Besoin 33 : Internationalisation

Description : L'application supportera le français, l'anglais et l'arabe . Les traductions seront stockées dans des fichiers JSON et l'utilisateur pourra basculer entre les langues via un sélecteur dans l'interface. L'interface doit prendre en charge l'écriture de droite à gauche (RTL) pour l'arabe afin d'assurer une expérience utilisateur correcte.

Besoin 34 : Design responsive

Description : L'interface client s'adaptera automatiquement aux différentes tailles d'écran, permettant une utilisation confortable sur ordinateur et portable.

Besoin 35 : 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 36 : 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 37 : 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 38 : 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 36.
- Navigateur(s) web sur lesquels votre client a été testé incluant la version de celui-ci, voir Besoin 37.
- Instructions pour configurer et lancer le serveur.
- Instructions pour installer et exécuter le client, y compris la configuration des dépendances (par exemple, via `npm install`).

Besoin 39 : Structuration du code

Description : Le code de l'application est organisé de manière claire et modulaire pour faciliter la maintenance et l'extensibilité :

- **Frontend** : Les composants Vue.js est organisés dans le dossier `frontend/src/components`, avec des fichiers comme `Edit.vue` (édition d'images), `Gallery.vue` (affichage des galeries), `Login.vue` (authentification), etc. Les fichiers TypeScript pour la logique API est placés dans `frontend/src`, comme `http-api.ts` pour les appels au serveur. Les traductions sont stockées dans `frontend/src/locale` (les fichiers `fr.json`, `en.json`, `ar.json`).
- **Backend** : Les classes Java sont organisées dans `backend/src/main/java/pdl/backend` avec des sous-dossiers comme `Image` (gestion des images, comme, `ImageController.java`, `ImageDao.java`), `Processing` (traitement d'images, par exemple, `Traitement.java`), `User` (gestion des utilisateurs, les fichiers, `UserController.java`), et `Security` (authentification, par exemple, `PasswordService.java`). Les ressources (comme les images) sont stockées dans `backend/src/main/resources/images`.