1 Introduction

Ce document présente les besoins nécessaires au développement d'une application de traitement d'image avec une architecture de type client-serveur. 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 et consiste principalement à articuler le contenu développé durant les premières semaines au sein des TP communs.

Le code de cette partie fera l'objet d'un rendu intermédiaire (29 mars).

Extensions : Des suggestions d'extensions seront proposées en cours (répartition des charges entre client et serveur, amélioration de l'interface utilisateur, traitements d'image plus avancés, généricité des algorithmes).

Le rendu final du projet est fixé au 16 avril.

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 en niveau des gris et en couleur enregistrées aux formats suivants :

— `JPEG`
— `TIF`

Lien ssh : git@gitlab.emi.u-bordeaux.fr :aderoo001/PDL-groupe5.git
Tag : Version1_delamorkitutou

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 enregister toutes les images présentes à l'intérieur du dossier `images`. Ce dossier `images` doit exister à l'endroit où est lancé le serveur. Le serveur doit analyser l'arborescence à l'intérieur de ce dossier. Seuls les fichiers image correspondant aux formats d'image reconnus doivent être 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 au moins 2 niveaux de profondeur dans l'arborescence. Le dossier contiendra 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 stocke les données brutes de chaque image ainsi que les 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).

Besoin 3: Appliquer un algorithme de traitement d'image

Description: Le serveur contient l'implémentation des algorithmes de traitement d'image proposés à l'utilisateur (voir partie 2.4). Dans le premier rendu on attend une implémentation uniquement pour les images couleur.

2.2 Communication

Pour l'ensemble des besoins, les codes d'erreurs à renvoyer sont précisés dans le paragraphe "Gestion d'erreurs".

Besoin 4: 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. `long`

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

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

Size : Une description de la taille de l'image (ex. 640*480*3 pour une image en couleur). `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.
Commentaire: Avec le besoin 2 la taille était sauvegardé comme le nombre total de pixel présent dans le l'image, de ce fait le choix du type d'algorithme à été décidé selon son extension. Or suite au besoin 4 le façon dont la taille est enregistré a changé, il aurait était peut-être plus intéressant de choisir le type d'algorithme selon le nombre de canaux présent dans sa taille.

Besoin 5: Ajout d'image

Description: L'envoi d'une requête `POST` à l'adresse `/images` au serveur avec des données de type multimedia dans le corps doit ajouter une image à celles stockées 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 (ex. PNG).

Besoin 6: Récupération d'images

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`.

Rendu-2: On a apporté une modification à la fonction getImage présente dans la classe ImageController. Dans sa version précedente nous avions besoin des 3 paramètres : algorithm, opt1 et opt2. Maintenant nous utilisons le type Map, qui contient toujours les mêmes informations mais regroupés dans un objet facile d'utilisations et bien plus propres. on a pu remplacer les nom opt1 et opt2, qui n'était pas très parlant, par des nom d'options bien plus compréhensible.

Besoin 7: Suppression d'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).
Gestion d'erreurs:

200 OK : L'image a bien été effacée.

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

Besoin 8: Exécution d'algorithmes par le serveur

Description: L'envoi d'une requête `GET` à une adresse de la forme `/images/id?algorithm=X&p1=Y&p2=Z` doit permettre de récupérer le résultat de l'exécution de l'algorithme `X` avec les paramètres `p1=Y` et `p2=z`. Un exemple plus concret d'URL valide est : `/images/23?algorithm=increaseLuminosity&gain=25`
En cas de succès, le serveur doit renvoyer l'image obtenue après traitement.
Gestion d'erreurs:

200 OK : L'image a bien été traitée.

400 Bad Request : Le traitement demandé n'a pas pu être validé par le serveur pour l'une des raisons suivantes :
— L'algorithme n'existe pas.
— L'un des paramètres mentionné n'existe pas pour l'algorithme choisi.
— La valeur du jeu de paramètres est invalide.
Le message d'erreur doit clarifier la source du problème.

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

500 Internal Server Error : L'exécution de l'algorithme a échoué pour une raison interne.

2.3 Client

Les actions que 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 9: 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.
Commentaire: Nous avons choisi de développer un carroussel.
Rendu-2: Une galerie d'image est maintenant display sur la droite, l'ancienne vue n'est plus accessible.

Besoin 10: Sélectionner une image et lui appliquer un effet

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 et choisir un des traitements d'image disponibles. Il peut être amené à préciser les paramètres nécessaires au traitement choisi (voir partie 2.4). L'image après traitement sera alors affichée sur la page.
Commentaire: Les métadonnées s'affichent au survol de l'image. Une option à été ajouté pour pouvoir performer l'image avec l'algorithme de niveaux de gris.
Rendu-2: Les métadonée sont maintenant afficher en permanence. Les images sont maintenants selectioné à partir des images affiché sur la gauche.

Besoin 11: Enregistrer une image

Description: L'utilisateur peut sauvegarder dans son système de fichier l'image chargée, avant ou après lui avoir appliqué un traitement.

Besoin 12: 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 n'est pas persistant (il n'y a pas d'ajout de fichier côté serveur).
Commentaire: Pour des raisons de confort nous avons décidé de refresh la vue à chaque ajout sur le backend.

Besoin 13: Suppression d'image

Description: Le client peut choisir de supprimer une image préalablement sélectionnée. Elle n'apparaitra plus dans les images disponibles sur le serveur.
Commentaire: Pour des raisons de confort nous avons décidé de refresh la vue à chaque suppression sur le backend.

2.4 Traitement d'images

Besoin 14: Réglage de la luminosité

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

Besoin 15: Égalisation d'histogramme

Description: L'utilisateur peut appliquer une égalisation d'histogramme à l'image sélectionnée. L'égalisation sera apliquée au choix sur le canal S ou V de l'image représentée dans l'espace HSV.
Commentaire: Ce besoin n'est pas effectif, il se trouve dans une errerur dans l'algorithme.
Rendu-2: Ce besoin est effictif à présent.

Besoin 16: Filtre coloré

Description: L'utilisateur peut choisir la teinte de tous les pixels de l'image sélectionnée de façon à obtenir un effet de filtre coloré.

Besoin 17: 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.

Besoin 18: Filtre de contour

Description: L'utilisateur peut appliquer un détecteur de contour à l'image sélectionnée. Le résultat sera issu d'une convolution par le filtre de Sobel. La convolution sera appliquée sur la version en niveaux de gris de l'image.
Rendu-2: La modification mentionné dans le mail à été ajouté.

Besoin 19: Filtre de niveau de gris ! ! !Exclusive

Description: L'utilisateur peut appliquer un filtre pour performer l'image en niveaux de gris.

2.5 Besoins non-fonctionnels

Besoin 20: Compatibilité du serveur

Description: La partie serveur de l'application sera écrite en Java (JDK 11) avec les bibliothèques suivantes :
org.springframework.boot : Version 2.4.2
net.imglib2 : Version 5.9.2
io.scif : Version 0.41
Son fonctionnement devra être éprouvé sur au moins un des environnement suivants :
— Windows 10
— Ubuntu 20.04
— Debian Buster
— MacOS 11

Besoin 21: Compatibilité du client

Description: Le client sera écrit en JavaScript et s'appuiera sur la version `3.x` du framework `Vue.js`.
Le client devra être testé sur au moins l'un des navigateurs webs suivants, la version à utiliser n'étant pas imposée.
— Safari
— Google chrome
— Firefox

Besoin 22: 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 20.
- Navigateur(s) web sur lesquels votre client a été testé incluant la version de celui-ci, voir Besoin 21.

2.6 Rendu version 2

Besoin 23: Propreté des sources

Rendu-2: Nous avons redigé la documentation afin que le parcours des fonctions sois plus simple pour un utilisateur, on a essayer d'organiser le code afin qu'ils sois plus orienté objet, On a nettoyé le code : on a mis des noms plus adéquats au méthodes et on a réduit la redondance de code en placant ces morceaux dans des classes appropriés ainsi que de nouveaux packages

2.6.1 Backend

Besoin 24: Liste d'algorithme

Description: Idée présente mais non implementée car non fonctionnelle.
Idée : l'idée et de construire une méthode dans le ImageController qui avec une liste d'algorithmes passé en paramètres, éxécute ceux ci sur la même image. L'intérêt est que si on souhaite ajouter un traitement d'image qui utilise des traitements déjà présent, on a pas besoin d'aller écrire quelque part dans le backend quelque ligne pour écrire une méthodes qui en appelle d'autres (mises à part rajouter l'option dans le frontend). A termes il serais aussi possible de remplacer chaque appel qui passe par getImage vers cette fonction car la liste contenant une seule Map revient à faire la même chose. On pourrais aussi proposé à l'utilisateur une option ou il peut enchainer lui mêmes les traitements qu"il souhaite. Implémentation : On a presque implémenté la fonction en question mais un problème fais qu'elle n'est pas fonctionnelle. on peut la retrouver dans ImageController méthode runCustomAlgorithm. On a eu l'idée d'utiliser un ArrayList<Map>String, String» enregistré dans un JSON. Nous nous sommes inspirés de cette ressources : https ://www.baeldung.com/spring-mvc-send-json-parameters C'est ce à quoi sers la class CustomImageProcessingAlgorithm Malheureseument nous n'avons jamais réussi à faire marché l'idée de la liste. On a essayé avec un type Map<String, String> mais en vain. On c'est rabattu à une simple map (la version actuelle pour voir si l'idée était faisable, ce qui semble être le cas On peut la tester avec la commande suivante : curl -i -H "Accept : application/json" -H "Content-Type : application/json" -X POST –data '{"algorithmList" : {"algorithm" : "grayLevel"}} ' "http ://localhost :8080/images/custom/1"

2.6.2 Traitement d'image

Besoin 25: Preprocessing

Description: Le principe du preprocessing est d'appliquer les différents filtres de traitement d'images à l'image mais avec un coup en ressources plus faible. Pour ce faire, on peut redimensionner les images plus petites pour gagner en temps de calcul. A ce niveau, certains problèmes ce posent, notamment lors de l'application du filtre de flou, le rendu n'est pas représentatif du resultat attendu sur l'image originel.
Commentaire: Cet ajout n'a pas été implémenté, nous avons pensé à implémenter dans un nouveau package une classe utilisant la méthode getScaleInstance de la bibliothèque BufferedImage et appliquant nos traitements d'images au résultat.

Besoin 26: Parallelism

Description: Le parallelism a pour but d'améliorer les performances du serveur de traitement d'images. On divise une image et on applique les différents traitements d'images aux différentes parties de l'image fractionnée.
Commentaire: Cet ajout n'a pas été implémenté, cependant certaine piste de réflexion ont été suivis. Certains problèmes se posent avec cette méthode pour appliquer le parallélisme. Lors de l'application du flou, l'algorithme utilise les voisins d'un pixel pour calculer la valeur du pixel. Cependant si on découpe l'image, on ne dispose plus des voisins qui n'appartiennent pas à la fraction.

Lien ssh : git@gitlab.emi.u-bordeaux.fr :aderoo001/PDL-groupe5.git Tag :