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 3 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`

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.

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`.

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.

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.

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).

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.

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.

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.

2.5 Besoins non-fonctionnels

Besoin 19: 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 20: 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 21: 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 19.
— Navigateur(s) web sur lesquels votre client a été testé incluant la version de celui-ci, voir Besoin 20.

3 Implémentation des extensions

3.1 Nouvelles implémentations coté serveur (Backend)

Besoin 22: Filtre conversion couleur en gris

Description: Un filtre qui transforme une image de couleur en niveau de gris, cela fonctionne correctement en choisissant l'image que l'on veut appliquer le filtre puis appuyer sur "Apply gray filter" sur le tableau de liste deroulante.

Besoin 23: 3 filtres de couleur rouge, vert, bleu

Description: Nous avons 3 filtres de couleur rouge, vert et bleu, l'utilisation est la même que celui du filtre de gris. (A noté que pour les images de très grandes tailles le temps de calcul peut être long).

Besoin 24: Filtre pixelisation

Description: Nous avons implémenté un filtre qui pixelise l'image selectionné nous avons borné les paramètres pour évité un temps de calcul trop long.

3.2 Nouvelles implémentations coté client (Frontend)

Besoin 25: Epuration et amélioration de l'interface graphique de l'utilisateur

Description: Nous avons supprimé tous les informations inutiles comme par exemple le logo Vuejs et les directions Home et About puis nous avons ajouté un fond de page fixe et un logo à notre projet. Nous avons ajouté un "tableau" liste deroulante qui contient les filtres gris, rouge, bleu et vert. Les images selectionnées sont tous de tailles fixes, elles sont étirées en longueur et en hauteur

Besoin 26: Une page suplémentaire spécifique pour les Besoins nouveaux et un lien pour notre gitlab cremi

Description: Nous avons ajouté une page annexe où il y a tous les nouveaux besoins que nous avons ajoutés. Vous devez cliqué sur "Les Besoins" pour aller sur la page et cliquer sur "Revenir" pour revenir sur la page principale. Vous pouvez donc soit lire les Besoins dans le README.md sur le gitlab du cremi soit le lire sur la page annexe dédiée ou soit sur le fichier pdf écrit en Latex à la racine de notre projet.

Besoin 27: Gestion des erreurs

Description: Sur certains algorithmes où il faudra mettre un paramètre, il y aura une alerte/pop-up et un message correspondant à l'erreur (Si le paramètre est grand ou petit ou pas de paramètre du coup par exemple).

Besoin 28: Un bouton pour revenir sur l'image original

Description: Nous avons ajouté un bouton pour revenir sur l'image original après application d'un filtre.