SAE Lego – Documentation du backend Java

1. Introduction

Ce document présente la partie Java de la SAE « Img2Brick ».
Le rôle du backend Java est :

- de traiter les images fournies en entrée (redimensionnement, matrice de pixels),
- d'interagir avec l'usine Lego via son API REST,
- d'intégrer l'algorithme de pavage écrit en langage C,
- d'exposer une interface en ligne de commande simple pour les tests et l'interaction avec le PHP.

Tout le code Java est écrit en anglais (identifiants et commentaires) comme demandé dans le sujet.
Ce document explique l'architecture globale, les choix d'implémentation, les difficultés rencontrées, les limites et les améliorations possibles.

2. Architecture globale

L'organisation du code Java est structurée par responsabilités dans différents packages.

fr.uge.lego.app
Contient le point d'entrée LegoBackendMain. Cette classe gère la ligne de commande avec trois actions principales :

- resize : redimensionner une image en utilisant l'un des algorithmes d'interpolation,
- ping-factory : tester la connexion avec l'usine Lego,
- solve-challenge : récupérer un défi de facturation, résoudre la preuve de travail et envoyer la réponse.

fr.uge.lego.image
Partie traitement d'image :
ResolutionChanger : interface qui représente une stratégie de redimensionnement
NearestNeighborResolutionChanger : algorithme du plus proche voisin
BilinearResolutionChanger : interpolation bilinéaire sur un voisinage 2x2
BicubicResolutionChanger : interpolation bicubique simple sur un voisinage 4x4
ColorUtils : utilitaires pour les calculs de pondération

Ce découpage permet de séparer l'interface de la logique algorithmique, ce qui facilite l'ajout de nouvelles stratégies ou des benchmarks.

fr.uge.lego.brick
Modélisation des briques Lego indépendamment de l'usine :
BrickType : dimensions géométriques
BrickColor : nom de couleur et code RGB
Brick : brique produite par l'usine avec numéro de série et certificat

fr.uge.lego.stock
Gestion du stock disponible :
StockItem : type, couleur, quantité, prix unitaire
StockRepository : abstraction pour la gestion du stock
InMemoryStockRepository : simple implémentation en mémoire utilisée pour les tests et pour fournir les données au programme C

fr.uge.lego.paving
Connexion avec le programme C :
PixelMatrix : matrice immuable représentant les pixels de l'image finale
PlacedBrick : brique placée dans le pavage final
PavingSolution : solution de pavage, avec liste de briques, score de qualité et prix total
PavingEngine : interface abstraite pour différents moteurs de pavage
CProgramPavingEngine : implémentation qui appelle le programme C pavage_v3

fr.uge.lego.factory
Client HTTP pour communiquer avec l'API de l'usine :
FactoryConfig : configuration (base URL, email, clé secrète)
LegoFactoryClient : wrapper autour de HttpURLConnection, avec les requêtes :
ping
billing balance
billing challenge
billing challenge-answer
catalog
ordering quote-request
ordering order
ordering deliver

Les DTO contenus dans fr.uge.lego.factory.dto correspondent aux objets JSON renvoyés par l'API.
ProofOfWorkSolver implémente la résolution du challenge SHA 256 avec un compteur big-endian.

3. Intégration Java C

Le programme C pavage_v3 reçoit :
pieces.txt
image.txt
fichier de sortie pavage.out

Et écrit sur la sortie standard une ligne contenant :
chemin_sortie prix_total qualite_totale ruptures

La classe CProgramPavingEngine :

- reçoit PixelMatrix et une liste de StockItem,
- écrit deux fichiers temporaires respecter le format attendu par le C,
- appelle le programme pavage_v3 via ProcessBuilder,
- lit la première ligne non vide de la sortie standard,
- parse le prix et la qualité,
- crée un objet PavingSolution

Pour l'instant la liste de briques placées est vide, car aucun parseur de pavage.out n'a été implémenté. Cela suffit pour la première version, mais pourra être amélioré.

Format des fichiers générés :

image.txt
W H
puis H lignes contenant W codes RGB hexadécimaux

pieces.txt
id w h r g b price stock

L'alignement des formats entre le C et le Java a été soigneusement respecté.

4. Interaction avec l'usine Lego

FactoryConfig lit les variables d'environnement LEGOFACTORY EMAIL et LEGOFACTORY SECRET.
Si elles ne sont pas présentes, une exception est levée pour éviter les erreurs silencieuses.

LegoFactoryClient permet d'appeler l'API :
ping
getBalance
fetchBillingChallenge
sendBillingChallengeAnswer
getCatalogRaw
requestQuote
confirmOrder
requestDelivery

Les réponses JSON sont sérialisées avec Gson.

ProofOfWorkSolver résout le défi cryptographique du serveur en construisant des tableaux d'octets et en testant leurs empreintes SHA 256.

5. Compilation Maven

Le fichier pom.xml inclut :
Java 17
dépendance Gson version 2.11.0
plugin pour générer un jar exécutable avec LegoBackendMain comme main class

Commandes principales :
mvn compile
mvn package

6. Difficultés rencontrées

Aligner les formats de fichiers pour la communication Java C :
le risque était que le C et le Java interprètent différemment les mêmes champs.
La difficulté a été résolue en centralisant les méthodes writeImageFile et writePiecesFile.

Gestion des processus externes dans Java :
ProcessBuilder demande une bonne gestion des fichiers temporaires, flux d'entrée et sortie, et codes de retour.
Solution : rediriger stderr dans stdout, ne lire que la première ligne non vide, traiter tout code non nul comme une erreur.

Résolution de la preuve de travail :
problèmes potentiels sur l'endianness. Cela a été résolu en choisissant une convention big endian claire et centralisée.

7. Limitations et bugs connus

- Le fichier pavage.out n'est pas encore parsé, la liste de briques de PavingSolution reste vide.
- Pas d'accès direct à une base de données depuis Java, uniquement un stock en mémoire.
- La validation des données en entrée est minimale.
- La résolution de preuve de travail est bloquante et mono-thread.

8. Améliorations possibles

- Ajouter un parseur pour lire pavage.out et reconstruire la liste de PlacedBrick.

- Ajouter d’autres algorithmes de redimensionnement ou des heuristiques de couleur.
- Créer une implémentation JDBC pour lire le stock depuis PostgreSQL.
- Ajouter des tests unitaires.
- Exposer les fonctionnalités par API REST ou connexion PHP.

9. Conclusion

Le backend Java propose :

- une architecture claire et modulaire,
- un client API fonctionnel pour l’usine,
- un lien opérationnel avec l’algorithme C,
- une configuration Maven prête à compiler et à empaqueter.