**Joan Favrichon**

**Titre complet du cours**

**420-556-AL, groupe 2**

**TP Final.**

**Travail présenté à**

**Guillaume Courtemanche**

**Département de nom du département**

**Cégep André-Laurendeau**

**27/09/2024 :23h59**

**Signature de l’étudiant : Joan Favrichon** Veille Technologique

Définition du projet

Je souhaite développer une application web de galerie d’images, permettant d’en apprendre plus sur le langage Rust que j’ai récemment voulu apprendre. Le but est de comparer Rust avec une technologie appropriée pour réaliser ma mon application de galerie. Mon niveau en Rust est faible alors je profite de ce projet pour en apprendre davantage.

Fonctionnalités voulues

* Gestion des images :
* Téléversement, chargement, et sauvegarde.
* Transformations : redimension, rotation.
* Ajustements : luminosité.
* Gestion des données :
* Opérations CRUD en lien avec chargement, et sauvegarde des images et les plus importantes (Créer, Lire).
* Requêtes d’optimisation (tri, filtrage, pagination).
* Frontend interactif :
* Interface utilisateur avec React.
* Affichage de galeries d’images, téléversement et gestion via API backend.

Besoin technologique

J'aurai besoin d'une technologie capable de gérer, sauvegarder et lire des images, ainsi que d'une solution pour gérer une base de données. Il me faudra également une bibliothèque dédiée à la manipulation des images côté frontend et une autre pour permettre à Rust d'interagir avec React. Enfin, une technologie sera nécessaire pour établir l'interaction entre backend et la base de données. ￼

Tests comparatifs :

**Les tests sont dans git, dans le répertoire :**

Critères d’adéquation :

* Chargement, sauvegarde, et téléversement d’images.
  + Framework ImageIO et Image
  + Charger une image depuis un système de fichiers, pour la traiter ou l’afficher.
  + Évaluer la capacité à charger une image depuis un système de fichiers.
* Transformations et ajustements (rotation, luminosité).
  + Framework image, le même si possible
  + Appliquer des transformations comme la rotation
  + Redimensionner une image
  + Modifier la luminosité d’une image.
* Base de données PLSQL
  + Création des tables dans la base de données
  + Insertions dans une table
  + Lecture dans une table pour récupérer des données
* Intégration avec React pour fournir une interface utilisateur.
  + Framework pour interagir entre Rust et React
  + Création et gestion d'API pour l'interaction frontend-backend
* Différence entre Java est Rust pour le langage.
  + Barrel / import / crate
  + Gestion de la mémoire
* Tri et filtrage de données

Identification des technologies à comparer

Rust: Diesel et Image

* **Diesel** : Framework ORM pour gérer les interactions avec une base de données relationnelle (PLSQL). Sécurisé grâce à son typage fort, et accessible par Rust grâce au Framework Diesel.
* **Image** : Bibliothèque de manipulation d’images offrant des fonctionnalités variées comme le redimensionnement et l’ajustement des paramètres visuels.

Java : Spring Data et ImageIO

* **Spring Data** : Solution intuitive pour gérer les bases de données, compatible avec PLSQL, déjà pratiquer plusieurs fois.
* **ImageIO** : API standard de manipulation d’images, intégrée au JDK. Idéal pour des tâches simples, mais moins performante sur des transformations complexes.

Tests

**[Test] Backend (Rust et Java)**

**Java – ImageIO / Spring.**

Les tests sont tous dans le fichier org.exemple.testJava, se sont des mains exécutables individuellement car Spring n’accepte pas plusieurs mains de base. Ne pas hésiter à faire des mvn clean install si une erreur se présente à ce sujet. Il se peut donc que des tests se chevauche dans la console, c’est aussi à cause du fait que j’appelle dans chaque main le testJavaApplication pour instancier au moins une donnée dans la base de données.

**Test pour les images.**

|  |  |
| --- | --- |
| **Éléments validés** | Changement de la luminosité sur une image |
| **Procédure d’exécution** | Exécuter la main dans la classe TestLuminositer, ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestLuminositer" |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base avec moins de luminosité * Ouverture du fichier : imageBrightnessAdjusted.jpg |

|  |  |
| --- | --- |
| **Éléments validés** | Redimensionnement d’une image |
| **Procédure d’exécution** | Exécuter la main dans la classe TestRedimensionnement, ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestRedimensionnement" |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base mais plus grande * Ouverture du fichier : imageResized.jpg |

|  |  |
| --- | --- |
| **Éléments validés** | Rotation d’une image |
| **Procédure d’exécution** | Exécuter la main dans la classe TestRotationImage, ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestRotationImage" |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base mais tourner à 90 degrés * Ouverture du fichier : imageRotated.jpg |

**Test pour une connexion à React**

|  |  |
| --- | --- |
| **Éléments validés** | La connexion entre le backend et front-end est valide, les données peuvent circuler |
| **Procédure d’exécution** | Exécuter la main dans la classe TestReact, ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestReact" -X  Se déplacer vers le fichier fetest et exécuter react avec la commande :  npm start |
| **Résultats** | * Le programme se lance correctement * Le test ‘Hello from Java’ est bien afficher * Le bouton envoie bien une requête post. Checker dans l’inspecteur google pour vérifier la réponse de la requête |

**Test pour la base de données**

|  |  |
| --- | --- |
| **Éléments validés** | L’insertation d’une image dans la base de données, et la selection de cette instance. |
| **Procédure d’exécution** | Exécuter la main dans la classe TestJavaApplication  , ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestJavaApplication" |
| **Résultats** | * L’image a bien été insérer en base de données * Les informations de l’image sont affichées dans la console * La table est bien recréée * Les fichiers de différents types sont insérés dans la base de données |

Aucun test n’est effectué pour la création d’une table. Spring s’en occupe tous seul à l’exécution d’une main. Il instancie les classes avec l’annotation @Entity a chaque exécution.

**Test tri et filtrage**

|  |  |
| --- | --- |
| **Éléments validés** | Récupération des données suivant la date, donc un tri du plus récent au plus vieux et un filtrage sur une date précise. |
| **Procédure d’exécution** | Exécuter la main dans la classe TestTriImage  , ou à la racine du projet, dans un git bash exécuter la commande  mvn exec:java -Dexec.mainClass="org.example.testjava.TestTriImage" |
| **Résultats** | * Les images sont bien triées * L’image filtrer à la date précise est bien la bonne image |

Les autres critères n’ont pas de test affilier, ce sont plus du code utiliser pour les autres tests.

Pour le chargement, la sauvegarde et le téléversement d’images, **ImageIO** remplit bien son rôle grâce à ses méthodes dédiées, utilisées dans tous les tests. Cependant, il s'avère limité dans mon cas, car il ne permet pas de redimensionner les images et ne propose pas de fonctionnalités pour d'autres opérations en dehors du chargement, de la sauvegarde et du téléversement.

Exemple :

ImageIO.*read*(imageFile);

ImageIO.*write*(outputImage, "jpg", outputFile);

De plus que je n’ai pas fait de test exclus pour enregistrer deux différents types d’image dans la base de données, Il est inclus dans le test TestJavaApplication. Il y a que 2 types d’image insérer

**Rust – Image /Diesel.**

Les tests sont tous dans le fichier source, se sont des mains exécutables individuellement. Ils fonctionnent avec la même base de données que les test java.

**Test pour les images.**

|  |  |
| --- | --- |
| **Éléments validés** | Changement de la luminosité sur une image |
| **Procédure d’exécution** | Exécuter la main dans la classe test\_baisse\_luminositer, ou à la racine du projet, dans un git bash exécuter la commande  cargo run --bin test\_baisse\_luminositer |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base avec moins de luminosité * Ouverture du fichier : ImageBrightness.jpg |

|  |  |
| --- | --- |
| **Éléments validés** | Redimensionnement d’une image |
| **Procédure d’exécution** | Exécuter la main dans la classe test\_redimension\_image, ou à la racine du projet, dans un git bash exécuter la commande  cargo run --bin test\_redimension\_image |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base mais plus grande * Ouverture du fichier : ppsteam\_resized.jpg |

|  |  |
| --- | --- |
| **Éléments validés** | Rotation d’une image |
| **Procédure d’exécution** | Exécuter la main dans la classe test\_rotation\_image, ou à la racine du projet, dans un git bash exécuter la commande  cargo run --bin test\_rotation\_image |
| **Résultats** | * Récupération, ouverture et modification d’un fichier image sans générer d’erreur * La génération du fichier est correcte et l’image correspond à celle de base mais tourner à 90 degrés * Ouverture du fichier : image\_rotated.jpg |

**Test pour une connexion à React**

|  |  |
| --- | --- |
| **Éléments validés** | La connexion entre le backend et front-end est valide, les données peuvent circuler |
| **Procédure d’exécution** | Exécuter la main dans la classe TestReact, ou à la racine du projet, dans un git bash exécuter la commande  cargo run --bin test\_connexion\_rust  Se déplacer vers le fichier rust\_test et exécuter react avec la commande :  npm start |
| **Résultats** | * Le programme se lance correctement * Le test ‘Hello from Java’ est bien afficher * Le bouton envoie bien une requête post. Checker dans l’inspecteur google pour vérifier la réponse de la requête |

**Test pour la base de données**

|  |  |
| --- | --- |
| **Éléments validés** | L’insertation d’une image dans la base de données, et la selection de cette instance. |
| **Procédure d’exécution** | Exécuter la main dans la classe main |
| **Résultats** | * L’image a bien été insérer en base de données * Les informations de l’image sont affichées dans la console * La table est bien recréée * Les fichiers de différents types sont insérés dans la base de données |

Aucun test n’est effectué pour la création d’une table. Spring s’en occupe tous seul à l’exécution d’une main. Il instancie les classes avec l’annotation @Entity a chaque exécution.

**Test tri et filtrage**

|  |  |
| --- | --- |
| **Éléments validés** | Récupération des données suivant la date, donc un tri du plus récent au plus vieux et un filtrage sur une date précise. |
| **Procédure d’exécution** | Exécuter la main dans la classe test\_tri\_filtre  , ou à la racine du projet, dans un git bash exécuter la commande  cargo run --bin test\_rotation\_image |
| **Résultats** | * Les images sont bien triées * L’image filtrer à la date précise est bien la bonne image |

Les autres critères ne font pas l'objet de tests unitaires spécifiques. Ils sont plutôt du code réutilisé dans d'autres tests. Image est plus pratique car elle fournit des méthodes prêtes à l'emploi pour gérer les fichiers et implémenter mes fonctionnalités.

Pour l’insertion des différents types d’image, c’est comme dans mes test java. Cela se fait avec seulement deux type mais cette fois ci dans le code main.

Pour les différences de code entre Java et Rust, peu de temps ma été accorder pour les faire, alors sur la base de mes recherches / expériences dans rust est java j’ai constaté :

Barrel / Import / Crate

* Java : Utilise des packages pour organiser les classes et les interfaces. Les imports sont utilisés pour inclure ces packages dans le code. Par exemple, import java.util.List;.
* Rust : Utilise des crates pour organiser le code. Les crates peuvent être des bibliothèques ou des exécutables. Les modules (mod) et les imports (use) sont utilisés pour inclure des parties de crates. Par exemple, use std::collections::HashMap;.

Gestion de la mémoire

* Java : Utilise un ramasse-miettes (garbage collector) pour gérer automatiquement la mémoire. Cela simplifie la gestion de la mémoire pour le développeur, mais peut introduire des pauses imprévisibles dans l'exécution du programme.
* Rust : Utilise un système de propriété et d'emprunt pour gérer la mémoire sans garbage collector. Cela permet un contrôle plus précis de la mémoire et des performances plus prévisibles, mais nécessite une gestion explicite par le développeur.

Ces différences reflètent les philosophies de conception de chaque langage : Java favorise la simplicité d'utilisation au prix de performances potentiellement moins prévisibles, tandis que Rust privilégie la sécurité et la performance au prix d'une complexité accrue dans la gestion de la mémoire.

Comparaison de l’adéquation générale des deux technologies

**Chargement, sauvegarde, et téléversement d’images**

* **Java (ImageIO)** :
  + **Chargement** : Utilise ImageIO.read(imageFile) pour charger des images. Fonctionne bien pour des tâches simples.
  + **Sauvegarde** : Utilise ImageIO.write(outputImage, "jpg", outputFile). Efficace pour les formats standards.
  + **Téléversement** : Gère bien les opérations de base mais manque de fonctionnalités avancées comme le redimensionnement.
* **Rust (Image)** :
  + **Chargement** : Utilise des fonctions de la bibliothèque image pour charger des images. Plus flexible et performant.
  + **Sauvegarde** : Utilise image.save(outputFile) pour sauvegarder des images. Supporte divers formats.
  + **Téléversement** : Offre des fonctionnalités avancées.

**Transformations et ajustements (rotation, luminosité)**

* **Java (ImageIO)** :
  + **Rotation** : Peut être réalisée mais nécessite des manipulations supplémentaires.
  + **Redimensionnement** : Limité, nécessite des bibliothèques supplémentaires.
  + **Luminosité** : Possible mais pas directement supporté par ImageIO, nécessite des manipulations d'image pixel par pixel.
* **Rust (Image)** :
  + **Rotation** : Supportée directement par la bibliothèque image.
  + **Redimensionnement** : Facilement réalisable avec des fonctions intégrées.
  + **Luminosité** : Supportée directement, permet des ajustements précis et performants.

**Base de données PLSQL**

* **Java (Spring Data)** :
  + **Création des tables** : Automatisée avec les annotations @Entity.
  + **Insertions** : Facile avec les méthodes de repository.
  + **Lecture** : Efficace avec les requêtes JPQL et les méthodes de repository.
* **Rust (Diesel)** :
  + **Création des tables** : Nécessite des migrations explicites mais offre un contrôle précis.
  + **Insertions** : Sécurisées et performantes grâce au typage fort mais nécessite plus d’effort avec l’écriture suivante : diesel::insert\_into
  + **Lecture** : Performante avec des requêtes SQL, mais aucune intégrer comme dans java.

**Intégration avec React**

* **Java (Spring Boot)** :
  + **Framework** : Utilise Spring Boot pour créer des APIs RESTful.
  + **Interaction** : Facile avec des annotations comme @RestController.
  + **Gestion d'API** : Bien supportée avec des outils comme Swagger.
* **Rust (Actix Web)** :
  + **Framework** : Utilise Actix Web pour créer des APIs RESTful.
  + **Interaction** : Performante mais nécessite plus de configuration.
  + **Gestion d'API** : Plus complexe à comprendre, mais dans le même principe qu’avec Spring.

**Différence entre Java et Rust pour le langage**

* **Barrel / Import / Crate** :
  + **Java** : Utilise des packages et des imports (import java.util.List;).
  + **Rust** : Utilise des crates et des modules (barrel =) je vais vous l’expliquer à la présentation promis)(use std::collections::HashMap;).
* **Gestion de la mémoire** :
  + **Java** : Utilise un garbage collector pour la gestion automatique de la mémoire.
  + **Rust** : Utilise un système de propriété et d'emprunt pour une gestion de mémoire sans garbage collector.

**Tri et filtrage de données**

* **Java (Spring Data)** :
  + **Tri** : Facile avec les méthodes de repository.
  + **Filtrage** : Efficace avec les requêtes JPQL.
* **Rust (Diesel)** :
  + **Tri** : Performant avec des requêtes SQL typées, moins pratique.
  + **Filtrage** : Sécurisé et précis grâce au typage fort.

**Conclusion**

* **Java** : Idéal pour des projets nécessitant une intégration rapide et une gestion simplifiée des bases de données et des APIs. Moins performant pour des manipulations d'images complexes.
* **Rust** : Offre des performances supérieures et un contrôle précis sur la mémoire et les opérations complexes, mais nécessite plus de configuration et de gestion explicite.

Options technologiques la plus appropriée pour le projet selon moi.

**Description des langages Java et Rust basées sur les tests**

Pour votre projet, bien que Rust offre des avantages en termes de performance et de manipulation d'images, **Java** semble être la meilleure option pour plusieurs raisons :

1. Simplicité et Familiarité : J’ai déjà accumulé pas mal d'expérience en Java, ce qui rendra le développement plus rapide et moins complexe. Les tests montrent que les opérations de base comme le chargement, la sauvegarde et le téléversement d'images sont bien gérées par ImageIO, même si ce dernier est limité pour des transformations plus complexes.
2. Compatibilité : Spring et les bases de données PLSQL sont bien intégrés, facilitant la gestion des données. Les tests ont démontré que Spring Data gère efficacement les opérations CRUD et les interactions avec la base de données.
3. Écosystème : Java dispose d'un écosystème mature et bien supporté, ce qui simplifie l'intégration avec des frameworks frontend comme React. Les tests ont confirmé que la connexion entre le backend Java et le frontend React fonctionne bien, avec des échanges de données fluides.
4. Manipulation d'images : Bien que ImageIO soit limité, les tests ont montré qu'il peut gérer des tâches de base comme le redimensionnement et la rotation avec des manipulations supplémentaires. Pour des besoins plus avancés, vous pourriez envisager d'utiliser une bibliothèque Java plus puissante comme Apache Commons Imaging ou TwelveMonkeys ImageIO. Même après tant de temps passer sur ce langage, les modifications d’image faite manuellement n’est pas si complexe après coup.

**Rust**

* Courbe d'apprentissage : Rust est un nouveau langage à apprendre, avec une première marche très haute. Les tests ont montré que bien que Rust offre des performances légèrement supérieures, il nécessite une gestion explicite et une configuration plus complexe.
* Compatibilité PLSQL : Diesel ne fournit pas toutes les fonctionnalités que Spring Data offre, rendant la gestion des bases de données moins intuitive. Les tests ont montré que bien que Diesel soit performant, il nécessite plus de configuration et de gestion explicite des migrations.
* Intégration avec React : L'intégration avec React est plus complexe en Rust, notamment en utilisant WebAssembly. Les tests ont montré que bien que possible, cette intégration nécessite plus de configuration et de gestion.

**Conclusion finale**

Pour un projet nécessitant des performances élevées et une gestion fine des ressources, Rust pourrait être envisagé. Cependant, compte tenu de mon expérience en Java, de la compatibilité avec les outils que vous utilisez déjà, et des résultats des tests, **Java** est la solution la plus simple et la plus appropriée pour ce projet actuel. Java va me permettre de développer plus rapidement et de manière plus efficace, tout en utilisant des outils et des frameworks avec lesquels vous êtes déjà familier.

Mon choix technologique

Pour ce projet, j'ai choisi d'utiliser Rust :

**Motivation Personnelle**

* **Apprentissage et Expérience** : Choisir Rust me permet d'apprendre un nouveau langage et d'acquérir de l'expérience dans un domaine différent. Cela enrichit mes compétences et ouvre de nouvelles opportunités professionnelles.
* **Défi** : Rust représente un défi stimulant. Sa courbe d'apprentissage est moins rapide que celle de Java, mais cela rend l'expérience d'autant plus gratifiante. Les tests ont montré que, malgré la complexité initiale, la compréhension de la connexion avec PLSQL et React est accessible.
* **Innovation** : En utilisant Rust, je me positionne à la pointe de l'innovation technologique. Rust est souvent choisi pour des projets nécessitant une grande fiabilité et performance, ce qui peut être un atout.

En conclusion, j'ai choisi Rust pour ce projet non seulement pour ses avantages techniques, mais aussi pour les opportunités d'apprentissage et de développement personnel qu'il offre. Bien que cela représente un défi, les bénéfices en termes de compétences acquises et de performance du projet en valent la peine. Rust est un langage moderne et puissant qui, malgré une documentation parfois moins abondante, permet de réaliser des projets robustes et performants.