Conception et implémentation d'un émulateur du service Textract de AWS

Rapport de stage

Jalal IZEKKI

jizekki@enseirb-matmeca.fr

Maître du stage : M. Abderrahmane Bagdouri
Encadrant du stage : M. Frédéric Herbreteau

25 octobre 2021

Département Informatique
Enseirb-Matmeca

Résumé

Amazon Web Services (**AWS**) est une plate-forme qui propose plus de 200 services variant entre le calcul, le stockage, le machine learning et l'analyse. **AWS** possède une communauté vaste et dynamique, avec des clients de pratiquement tous les secteurs et de toutes les tailles, y compris les startups, les grandes entreprises et les organisations du secteur public.

`Textract` est l'un des services de **AWS** les plus connus et les plus utilisés. Il permet d'extraire automatiquement du texte, de l'écriture manuscrite et des données (comme les tableaux et les formulaires) à partir des documents numérisés.

AWS utilise le principe du paiement à l'utilisation pour `Textract` comme pour plusieurs autres services ; le client paye à chaque utilisation d'un service. Pour les développeurs ayant besoin de développer et tester des applications intégrant ces services, ceci pourrait être très coûteux.

Une solution pour ce problème serait d'utiliser un émulateur des services de **AWS** en local pendant l'étape de développement et test, et ensuite passer sur l'environnement de **AWS** pour le déploiement de l'application développée. L'émulateur de chaque service disposera d'une interface similaire à celle que propose **AWS** pour ce même service. Aucun changement du code ne serait alors nécessaire en passant de l'émulateur à l'environnement **AWS**.

La réalisation de cet émulateur se fait avec le framework **Spring** de façon à avoir des émulateurs qui ne contiennent initialement qu'un seul service, et puis intégrer tous les émulateurs dans un seul et unique projet. L'objectif de mon stage consiste à concevoir, réaliser et tester l'émulateur du service `Textract`. La réalisation de cet outil a requis l'utilisation du moteur `Tesseract` pour la reconnaissance optique des caractères et de la bibliothèque `OpenCV` pour le traitement des images.

Table des matières

1 Introduction

1.1 Présentation de l'entreprise

OvalSquare est une start-up très récente. Elle a été créée en 2021 à Tétouan au Maroc (voir la figure 1) par 4 personnes : Amine Bagdouri (CEO & Product Owner), Ali Louzzi, Ahmed Kassi et Abderrahmane Bagdouri. La start-up a pour but d'innover dans le domaine informatique et rendre ses services accessibles à tous, notamment les services liés au *Cloud Computing*. C'est dans ce cadre que s'inscrit le premier projet ambitieux `Nephojar AWS Emulator` sur lequel s'est fondée la start-up. Celui-ci permettra aux clients d'utiliser un émulateur des services proposés par `Amazon Web Services` (**AWS**) afin de développer et tester leurs applications dans un environnement local sans coûts supplémentaires avant de déployer sur un vrai environnement de production **AWS**.

Il est possible de joindre le personnel de **OvalSquare** en envoyant un message à l'adresse mail `contact@ovalsquare.tech` ou en se rendant sur le site internet de la start-up : `ovalsquare.tech`.

FIGURE 1 – Localisation de la start-up **OvalSquare** sur la carte

1.2 Service occupé durant le stage

OvalSquare est une entreprise en phase de démarrage, composée avant tout d'une équipe ambitieuse et déterminée dont la mission principale est d'expérimenter de nouvelles solutions. En tant que stagiaire, la mission qui m'a été attribuée durant le stage est une mission de recherche et développement, il s'agissait de concevoir, développer et tester une solution informatique.

2 Mission technique et solution

2.1 Présentation du contexte

AWS est une plate-forme Cloud mondiale qui permet d'héberger et gérer des services sur internet. Elle possède la communauté la plus vaste et la plus dynamique du monde, avec des clients de tous les secteurs et de toutes les tailles. Les entreprises l'utilisent pour innover plus rapidement, en toute agilité et à moindre coût.

Les services proposés par **AWS** diffèrent entre des services de stockage Cloud et récupération des données (`Amazon S3`), le calcul évolutif dans le Cloud (`Amazon EC2`), la gestion et la transmission des messages et notifications aux utilisateurs (`Amazon SNS`), l'extraction et l'analyse automatiques du texte à partir des documents numérisés (`Amazon Textract`) ainsi que beaucoup d'autres services et fonctionnalités.

AWS offre des services technologiques à la demande avec une tarification à la carte ; on ne paye que ce que l'on utilise. Les développeurs se servent de ces services pour créer leurs applications, les tester et les exécuter en ligne. [1]

2.2 Problématique

L'environnement **AWS** agit comme une collection intégrée de services matériels et logiciels. Chaque service propose une interface de programmation d'applications (**API**), c'est le seul moyen qui permet d'interagir avec le service. Amazon continue toujours à améliorer ses services. Pour un développeur, il est donc impératif de se référer constamment à des guides de développeurs comptant des centaines de pages pour comprendre le fonctionnement des **APIs** de **AWS**. Ceci prête souvent à la confusion quand il s'agit de développer une application plus ou moins complexe. Un développeur doit donc également jouer avec les différentes **APIs** en passant des appels d'**API** plusieurs fois avec des paramètres différents, que ce soit pour assimiler le fonctionnement d'une **API**, pour l'intégrer dans son application et tester son bon fonctionnement ou pour détecter et corriger des anomalies techniques dans l'application développée.

Cependant, la plupart des services de **AWS** imposent une facturation à chaque utilisation, c'est-à-dire que chaque appel à une fonction d'une **API** implique des frais supplémentaires. Le travail de développement et test pourrait donc engendrer des frais importants, notamment si des bugs apparaissent dans l'application en cours de développement.

2.3 Solution proposée

Nephojar AWS Emulator est un projet qui a pour but de simuler le fonctionnement des services les plus utilisés de **AWS**. Et ce, afin de permettre aux développeurs d'applications de se passer de **AWS** pendant l'étape de développement et test. En utilisant un émulateur avec des **APIs** ayant les mêmes interfaces que **AWS**, les développeurs

1. Certaines de ces informations sont extraites de : `aws.amazon.com/fr/what-is-aws/`

peuvent effectuer des appels illimités en local et obtenir des résultats similaires à ceux de **AWS** tout en évitant d'engendrer des frais d'utilisation supplémentaires. Quand les développeurs sont satisfaits du fonctionnement de leur application, ils pourront la déployer pour fonctionner dans le Cloud avec les services de **AWS**.

Textract est un des services de **AWS** de plus en plus utilisé. Il permet de lire les documents comme le ferait une personne humaine, en extrayant du texte, des tableaux, des formulaires et d'autres données structurées sans configuration ni code personnalisé [2]. L'intégration de ce service dans le projet **Nephojar** s'est donc avérée une nécessité.

2.4 Présentation des outils utilisés

L'implémentation de la solution se fait en langage **Java**. L'application devrait exposer ses fonctionnalités sous forme d'un **service Web Restful** [3], l'utilisation du *framework* **Spring** semble donc être un bon moyen pour la réalisation.

Définition

Spring est un framework open source pour construire et définir l'infrastructure d'une application Java, dont il facilite le développement et les tests. Il se caractérise par une inversion de contrôle assurée la recherche et l'injection des dépendances ainsi que par une couche d'abstraction qui permet d'intégrer d'autres *frameworks*.

Pour la reconnaissance optique de caractères (**OCR**), plusieurs choix ont été étudiés en termes de précision, accessibilité et possibilité de configuration. Au final, `Tesseract` a été choisi étant le meilleur outil *Open Source* disponible à cet effet.

Définition

Tesseract est un logiciel qui permet d'effectuer la reconnaissance optique de caractères dans plusieurs langues (dont l'anglais et le français). C'est un outil gratuit développé en `C` et `C++` et dont les sources sont publiées sous licence Apache. Le développement de cet outil est actuellement poursuivi par Google.

`Tesseract` est écrit en `C/C++`, mais il existe un *wrapper* appelé `Tess4j` (*Tesseract for Java*) qui permet de l'utiliser en `Java`.

Afin d'analyser les images pour détecter les formulaires et tableaux, il faudra utiliser un outil de traitement d'images. La bibliothèque **OpenCV** semble être le meilleur choix étant *Open Source* et disposant de plusieurs fonctionnalités.

2. Certaines de ces informations sont extraites de : `https://aws.amazon.com/fr/textract/`
3. Un service Web Restful est une application qui expose entièrement ses fonctionnalités comme un ensemble de ressources identifiables par un `URI` et accessibles par la syntaxe du protocole `HTTP`. Pour plus d'informations `https://fr.wikipedia.org/wiki/Service_web`

Définition

OpenCV (**Open Computer Vision** en anglais) est une bibliothèque graphique qui met à disposition de nombreuses fonctionnalités permettant le traitement des images et des vidéos, la réalisation des interfaces graphiques et le calcul matriciel. Elle dispose également de l'implémentation de certains algorithmes classiques de l'apprentissage artificiel.

En outre, la bibliothèque `pdfbox` a été utilisée pour manipuler les fichiers **PDF**.

3 Réalisation de la solution

3.1 Structure du projet

Le projet de l'émulateur du service `Textract` prend le format d'un projet **Spring**. Il est composé de plusieurs paquets (*packages*) comme le montre la figure 6 des annexes. L'émulateur se compose de cinq couches principales (Cf. figure 2). Les parties suivantes décrivent le rôle de chacune de ces couches.

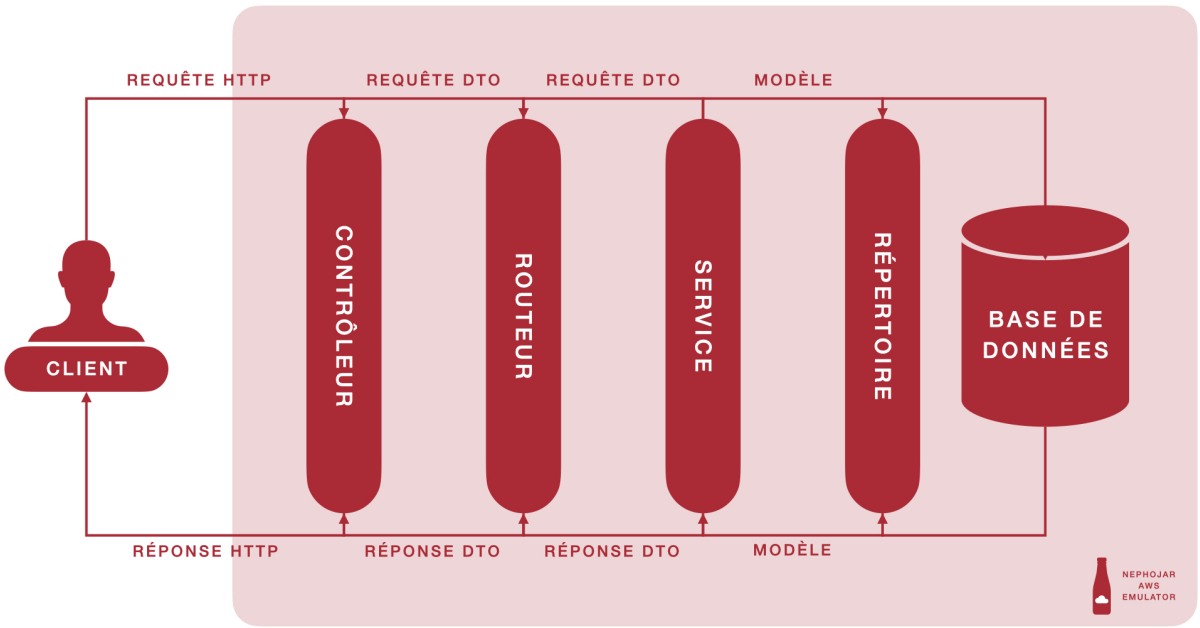

FIGURE 2 – Structure du projet Nephojar

3.1.1 Le contrôleur

Le contrôleur donne accès au comportement de l'application et définit son interface. La classe qui définit le contrôleur est précédée par l'annotation `@RestController` qui

indique qu'il s'agit bien du contrôleur et que les données renvoyées par toutes ses méthodes seront écrites directement dans le corps de la réponse `HTTP`. Le contrôleur contient une méthode pour chacune des opérations qu'offre l'**API** du service. Il contient en plus un attribut `router` qui permet de rediriger les requêtes vers la bonne opération.

Le contrôleur a également été annoté par `@RequestMapping(header="textract")` pour éviter que celui-ci traite des requêtes relatives à d'autres services (dont le `header` ne commence pas par `textract`).

En plus du contrôleur, **Spring** fournit un moyen très utile pour gérer les exceptions levées par la classe du contrôleur à l'aide d'une classe annotée par `@ControllerAdvice`. Cette classe contient une méthode pour chaque exception qui pourrait être levée lors de l'exécution de l'application. Chaque méthode enrichie l'entité de réponse avec des informations sur l'exception survenue avant de la renvoyer. Les exceptions qui peuvent être levées ont aussi été implémentées, elles permettent d'informer le client de la nature du problème survenu et de la raison qui y a donné lieu.

3.1.2 Le routeur

Le routeur est la deuxième couche de l'application, son rôle est de lier les différentes opérations avec le contrôleur. Initialement, un faux routeur a été implémenté pour faciliter le développement. Celui-ci a été progressivement écrasé par un vrai routeur. Les deux routeurs héritent d'une interface commune et sont tous les deux des composantes de **Spring**. Pour n'en utiliser qu'un seul à la fois, le vrai routeur a été annoté par `@Primary`, **Spring** devrait ainsi chercher des redirections dans le vrai routeur en premier, et s'il n'en trouve pas, il utilise celles du faux routeur.

3.1.3 Les services

Les services sont en fait les points de terminaison (*endpoints*) que propose l'**API** de `Textract`. Chaque point de terminaison dispose d'une classe contenant une méthode portant le même nom et dans laquelle on définit le comportement du service. Les points de terminaison dans l'**API** de `Textract` sont les suivants :

- `DetectDocumentText` : détecte le texte dans un document et renvoie la réponse au client. Il s'agit d'une opération synchrone.
- `StartDocumentTextDetection` : démarre la détection asynchrone de texte et renvoie un identifiant pour récupérer le résultat plus tard.
- `GetDocumentTextDetection` : obtient les résultats d'une opération asynchrone précédente qui détecte du texte dans un document.
- `AnalyzeDocument` : analyse un document d'entrée pour trouver les relations entre ses éléments (formulaires ou tableaux). Cette opération est synchrone.
- `StartDocumentAnalysis` : démarre l'analyse asynchrone d'un document d'entrée pour détecter les relations entre ses éléments et renvoie un identifiant qui permettra de récupérer le résultat plus tard.

- `GetDocumentAnalysis` : obtient le résultat d'une opération asynchrone précédente qui analyse le contenu d'un document.

Les détails de l'implémentation et du fonctionnement de ces opérations seront détaillés plus loin dans ce document.

3.1.4 Les répertoires

Les répertoires en **Spring** sont un mécanisme d'encapsulation du stockage, de l'extraction et de la recherche des données dans les bases de données. Un répertoire est une interface qui hérite de l'interface `JpaRepository<T, ID>` où le type `T` correspond au type de l'objet géré par le répertoire et le type `ID` correspond au type de l'attribut utilisé comme identifiant des objets `T`. Cette dernière interface fournit un ensemble de méthodes pour interagir avec une base de données. Seulement les éléments qui devraient être sauvegardés dans la base de données auront un répertoire.

3.1.5 La base de données

La base de données permettra de stocker les résultats des méthodes asynchrones pour une durée maximale de 7 jours afin de permettre au client de les récupérer quand il le souhaite. Un planificateur de tâches a été utilisé pour détruire tout résultat n'ayant pas été récupéré au delà de 7 jours, les résultats récupérés sont supprimés immédiatement.

Le système de gestion des bases de données utilisé est `PostgreSQL`. Cependant, toute communication avec la base de données, que ce soit pour la lecture ou l'écriture, s'effectue par l'intermédiaire des répertoires implémentés.

3.2 Mise en place des modèles et DTO

Avant de commencer l'implémentation des différents points de terminaison, il faut d'abord implémenter les modèles et les objets de transfert de données (**DTO**) qui seront utilisés par ces points de terminaison. Les modèles seront nécessaires pour le stockage des objets dans la base de données, notamment pour les opérations asynchrones, alors que les **DTOs** sont utilisés pour communiquer les données entre les services et le contrôleur.

Information

Dans les architectures multicouches, les données sont généralement représentées différemment à chaque couche. Ce qui signifie que la communication entre les couches peut devenir lourde. Cela peut être évité en exploitant le modèle **DTO** en définissant des classes simples pour transférer les données entre les couches, simplifiant ainsi la communication.

Il a aussi fallu mettre en place des classes avec des méthodes de mappage pour mettre en correspondance les **DTOs** et les modèles, c'est-à-dire pour transformer facilement les modèles en **DTO** et réciproquement.

3.3 Implémentation des différentes opérations

L'**API** de `Textract` propose six points de terminaison, dont deux pour la détection et l'analyse des documents en mode synchrone, deux autres pour lancer la détection et l'analyse en mode asynchrone et deux derniers points de terminaison pour obtenir les résultats des opérations asynchrones. La figure 3 représente le fonctionnement du service `Textract` et ses différentes fonctionnalités.

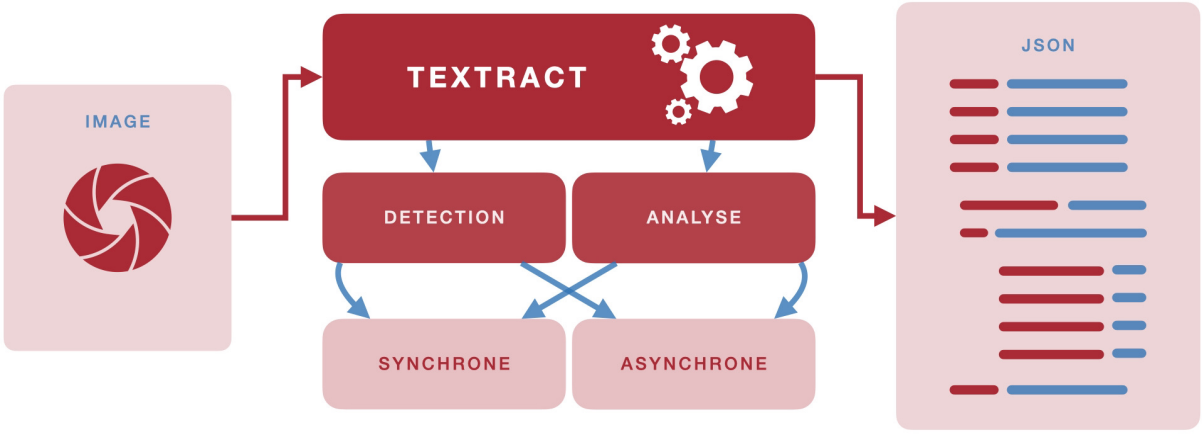

FIGURE 3 – Les différentes fonctionnalités du service Textract

Les opérations synchrones se caractérisent par leur rapidité et leur résultat est transmis directement au client comme réponse à sa requête. Les fichiers d'entrée ne peuvent donc être qu'une seule image. Tandis que les opérations asynchrones peuvent prendre en charge des documents de grande taille, notamment les fichiers **PDF**, la réponse envoyée au client lors d'une requête de ce type n'est donc qu'une clé qui lui permettra de récupérer le résultat de la détection ou analyse plus tard.

3.3.1 L'opération `DetectDocumentText`

Pour cette première opération, il a été important de prendre du temps pour comprendre le fonctionnement du service en entier ainsi que l'utilisation de l'outil `Tess4j`.

Pour commencer la reconnaissance optique avec `Tess4j`, il suffit de créer une instance de la classe `Tesseract` et spécifier le chemin vers le dossier où se trouvent les données utilisées pour l'entraînement du modèle.

```
ITesseract instance = new Tesseract();
instance.setDatapath("path/to/tessdata/folder");
```

Il est possible de changer quelques paramètres [4] de l'instance du moteur `Tesseract` en utilisant la méthode suivante :

4. Pour voir la liste de ces paramètres : `https://muthu.co/all-tesseract-ocr-options/`

```
instance.setVariable("variableName", "variableNewValue");
// en remplaçant variableName par le nom du paramètre à modifier et
// variableNewValue par la nouvelle valeur du paramètre.
```

Ensuite, on peut appeler la méthode suivante pour obtenir les mots détectés. Il est également possible d'obtenir une liste des lignes au lieu des mots :

```
List<Word> listOfWords, listOfLines;
listOfWords = instance.getWords(image, ITessAPI.TessPageIteratorLevel.RIL_WORD);
listOfLines = instance.getWords(image, ITessAPI.TessPageIteratorLevel.RIL_LINE);
```

La liste obtenue contient des objets `Word` dans lesquels on trouve les mots (ou lignes) détectés, leur position dans l'image d'entrée ainsi que la précision de la détection. Ces informations sont utilisées pour construire les blocs de la réponse. Celle-ci devrait être similaire à la réponse qui serait envoyée par **AWS** pour la même requête. Elle devrait donc contenir un bloc représentant l'image, celui-ci référence tous les blocs des lignes détectées. Chaque bloc de ligne fait référence aux blocs des mots qui composent la ligne.

3.3.2 L'opération `StartDocumentTextDetection`

Il s'agit de la première opération asynchrone à implémenter. Il a donc fallu avant tout mettre en place une méthode permettant de traiter les fichiers **PDF** et d'en extraire une liste d'images bufferisées. Chaque image sera ensuite traitée de la même façon que dans l'opération précédente. Cependant, au lieu de construire directement la réponse, les informations détectées sont stockées dans la base de données. Cette démarche est réalisée en asynchrone, **Spring** fournit l'annotation `@async` pour permettre de lancer une méthode en mode asynchrone (dans un nouveau thread).

Un identifiant est envoyé au client pour récupérer le résultat de la détection quand il le souhaite, dans la limite de 7 jours.

Au fur et à mesure de la détection, `Textract` doit publier des valeurs d'état dans une rubrique du service `Amazon SNS` pour indiquer au client l'état d'avancement de la détection. Les informations pour accéder à cette rubrique sont indiquées par le client dans sa requêtre.

3.3.3 L'opération `GetDocumentTextDetection`

Cette opération peut être exécutée plusieurs fois à la suite d'une requête de détection asynchrone. Et ce, jusqu'à l'obtention de l'intégralité des blocs détectés. Les conditions pour faire plusieurs fois cette requête sont les suivantes :

1. Le client peut à tout moment faire une requête pour obtenir le résultat d'une requête de détection précédente même si celle-ci n'est pas encore terminée. Dans ce cas, seulement ce qui a été détecté lui sera envoyé. Le client devra refaire une requête plus tard pour obtenir le reste des blocs détectés.
2. Le client peut spécifier un nombre maximal de blocs à ne pas dépasser dans la réponse. À défaut, celui-ci est fixé à 1000 blocs. L'opération de détection d'un fichier

volumineux peut produire une réponse avec un grand nombre de blocs pouvant dépasser 1000 blocs. En l'occurrence, le client peut continuer à envoyer la requête `GetDocumentTextDetection` jusqu'à l'obtention de la totalité des blocs.

Pour indiquer au client qu'il lui est toujours possible d'effectuer une nouvelle requête, un *token* lui est envoyé avec la dernière réponse. Ce *token* devrait être renvoyé par le client dans sa prochaine requête pour confirmer qu'il a bien reçu la réponse précédente.

Toutes ces contraintes ont montré qu'il fallait plus d'agilité dans la manière dont les données sont stockées dans la base de données. En fait, il a fallu mettre des curseurs pour indiquer les blocs qui n'ont pas encore été envoyés au client et supprimer ceux qui l'ont été de la base de données dès que le client confirme leur réception. Pour ce faire, un *wrapper* des données a été implémenté, celui-ci emballe les données ainsi que les curseurs. Il a également été nécessaire d'ajouter un attribut pour sauvegarder le *token* envoyé au client. En outre, Il a été important de contrôler les accès à la base de données pour éviter tout conflit quand les opérations `StartDocumentTextDetection` et `GetDocumentTextDetection` s'effectuent en parallèle.

3.3.4 L'opération `AnalyzeDocument`

Pour analyser un document (une image dans le cas d'une opération synchrone), il faut d'abord détecter le texte qui se trouve en dehors des tableaux en utilisant les méthodes implémentées pour l'opération `DetectDocumentText` et en construire des blocs, `Tess4j` peut facilement être paramétré pour réaliser cette opération. Ensuite, on peut commencer à détecter les tableaux. Pour cela, on utilise la bibliothèque `OpenCV`, celle-ci peut être incluse dans le projet en l'ajoutant au fichier des dépendances. Elle peut ensuite être importée dans le programme à l'aide de la ligne suivante :

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

L'objectif est de détecter les tableaux et les cellules qui les composent, déterminer la position des cellules dans les lignes et les colonnes du tableaux et ensuite extraire le texte de chaque cellule. Ceci peut être réalisé en suivant ces étapes :

1. La première étape consiste à importer l'image en niveaux de gris, et inverser ses couleurs. Ceci permettra de virer les pixels ayant une grande intensité pour séparer les objets d'intérêt (c'est-à-dire les contours des cellules qui composent le tableau) de l'arrière plan. Cette opération est réalisée à l'aide de l'algorithme de **Thresholding**. L'implémentation de celui-ci est fournie dans `OpenCV` et peut être utilisée de la façon suivante :

   ```
   Mat destination = new Mat();
   int thresholdType = Imgproc.THRESH_BINARY_INV | Imgproc.THRESH_OTSU;
   Imgproc.threshold(image, destination, 128, 255, thresholdType);
   ```

2. Ensuite, on définit deux noyaux qui comporteront les lignes horizontales et verticales de l'image. Ceci peut être fait à l'aide de la méthode `getStructuringElement` :

```
Size vSize = new Size(1, Math.floor(threshold.cols() / 100d));
Size hSize = new Size(Math.floor(threshold.cols() / 100d), 1);
Mat vKernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, vSize);
Mat hKernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, hSize);
```

3. Les matrices `vKrenel` et `hKernel` obtenues représentent des images qui ne contiennent que des lignes verticales et horizontales respectivement. Pour améliorer la définition de ces lignes, on peut facilement éroder et dilater plusieurs fois les images avec les deux méthodes `Imgproc.erode` et `Imgproc.dilate` respectivement.
4. Maintenant que les lignes sont bien définies, on combine les deux images et on inverse la couleur de l'image obtenue, la structure du tableau devient alors claire. Celle-ci permettra à `OpenCV` de tracer des contours sur l'image d'origine pour en extraire une image par chaque cellule avec la méthode `Imgproc.findContours`.
5. Les images obtenues sont classées selon leur position (en ligne et en colonne) dans le tableau. Chaque image est ensuite envoyée à l'algorithme de reconnaissance optique des caractères implémenté précédemment.
6. On construit ensuite un bloc pour chaque cellule, celui-ci dispose d'une liste des identifiants de tous les blocs des mots ayant été détectés dans la cellule. L'ensemble des blocs est ajouté à la réponse.

La détection des formulaires n'a pas été implémentée dans l'émulateur. Celle-ci aurait pu être réalisée avec le même principe (détection des contours des champs et des cases à cocher). **AWS** utilise un modèle basé sur l'apprentissage automatique pour détecter les formulaires. Une telle approche nécessiterait du temps pour rassembler une grande quantité de données et entraîner le modèle.

3.3.5 Les opérations `StartDocumentAnalysis` et `GetDocumentAnalysis`

Il s'agit des opérations utilisées pour lancer l'analyse asynchrone et récupérer son résultat. Ces opérations fonctionnent de la même façon que `StartDocumentTextDetection` et `GetDocumentTextDetection` et n'ont pas besoin d'implémenter du code supplémentaire pour sauvegarder les données dans la base de données car elles utilisent le même *wrapper* que les méthodes de détection asynchrone.

3.4 Gestion des dépendances

Le service `Textract` ne s'exécute pas indépendamment des autres services. En effet, pour accomplir certaines tâches, `Textract` a besoin d'interagir avec d'autres services de **AWS**. Voici quelques exemples de ces tâches :

- Pour pouvoir envoyer des requêtes, l'utilisateur de `Textract` doit disposer d'un ensemble de permissions. `Textract` vérifie ces permissions en consultant le service `Amazon IAM`.
- L'utilisateur peut envoyer une requête pour demander de détecter du texte dans un document stocké dans le service de stockage de **AWS**. Pour accéder au document, `Textract` interagit avec le service `Amazon S3`.

- Les objets sont chiffrés avant d'être sauvegardés dans la base de données. Le service utilisé pour créer et gérer les clés cryptographiques du chiffrement est `Amazon KMS`.
- Comme il a été indiqué plus haut, `Textract` utilise le service `Amazon SNS` pour publier des valeurs d'état dans des rubriques spécifiées par l'utilisateur.

Lors du développement, ces services n'étaient pas encore implémentés dans l'émulateur. Ne pouvant pas réaliser l'émulateur du service `Textract` sans ces services, il a fallu mettre en place un mécanisme pour simuler le fonctionnement de ces services. Plusieurs *mocks* ont alors été implémentés pour remplacer les services. Chaque *mock* dispose d'une interface minimale permettant d'utiliser le service simulé. Les valeurs de retour des méthodes de chaque *mock* ont été définies de façon à permettre de couvrir tous les cas d'utilisation possibles.

3.5 Test de la réalisation

3.5.1 Méthodologie de test

Pour assurer le bon fonctionnement de l'application, des tests unitaires ainsi que des tests de réalisation ont été implémentés. Les tests unitaires ont été principalement introduit pour tester certaines méthodes comme la fonction de génération des chaînes de caractères aléatoires (utilisée pour générer des identifiants pour les objets) ou encore la méthode utilisée pour transformer un fichier **PDF** en une liste d'images bufferisées.

D'autre part, les tests de réalisation permettent d'assurer que l'émulateur fonctionne de la même façon que le service `Textract` de **AWS**. La méthodologie utilisée est représentée dans la figure 4, elle consiste à envoyer deux requêtes pour chaque test, une à l'émulateur et l'autre à **AWS** et ensuite comparer les valeurs de retour, celles-ci peuvent être le nombre des mots détectés, leur position dans le document, la précision de détection ainsi que la distance de **Levenshtein** [5] entre les mots détectés par **AWS** et ceux détectés par l'émulateur.

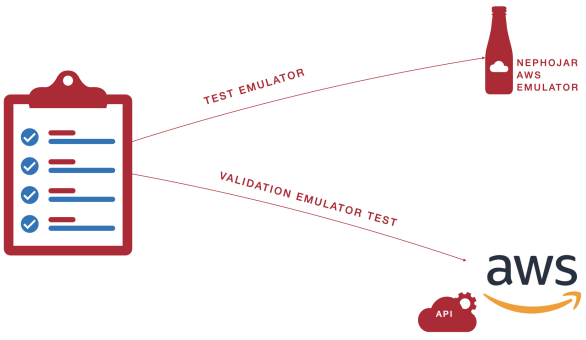

FIGURE 4 – Méthodologie de test de l'implémentation

5. La distance de **Levenshtein** mesure la différence entre deux chaînes de caractères en calculant le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre. Pour plus d'informations : https://fr.wikipedia.org/wiki/Distance_de_Levenshtein.

3.5.2 *Problèmes rencontrés*

Pour les fonctions asynchrones, lorsque la détection de texte est terminée, Textract devrait publier un état de complétion dans une rubrique spécifique d'un autre service, à savoir `Amazon SNS`. Pour obtenir les résultats de l'opération de détection ou analyse de texte, il faut d'abord vérifier que la valeur d'état publiée sur la rubrique `Amazon SNS` est `SUCCEEDED`. Ayant utilisé un *mock* pour les services `Amazon SNS` lors du développement, celui-ci n'est pas accessible au client, et donc aux tests, il n'était donc pas possible de savoir quand une opération asynchrone est terminée. Pour palier partiellement à ce problème, un autre *mock* a été réalisé au niveau des tests, il simule une attente suffisante pour terminer la détection avant de publier la valeur d'état `SUCCEEDED`.

4 Conclusion

4.1 Évaluation de la solution

À la fin du stage, une grande partie de l'émulateur du service `Textract` a été réalisée, il ne reste qu'à compléter l'algorithme de détection des formulaires et leur contenu. Pour cela, **AWS** utilise une intelligence artificielle et des techniques de l'apprentissage automatique pour détecter les formulaires dans les documents. Mais il pourrait être suffisant d'appliquer la même procédure de détection des tableaux avec certaines modifications dans le cas de l'émulateur. Ensuite, les opérations de l'analyse asynchrone des documents pourraient être finalisées sans aucune difficulté.

Les tests effectués montrent que l'émulateur donne de bons résultats et pourrait donc à la fin de la réalisation être utilisé par les développeurs pour tester leurs applications. Il pourrait cependant y avoir des différences entre les résultats de l'émulateur et ceux de **AWS** étant donné que les outils et techniques utilisés pour la reconnaissance optique des caractères ne sont pas les mêmes. À titre d'exemple, l'émulateur de `Textract` est moins puissant quand il s'agit de détecter ou analyser une écriture manuscrite. Ceci revient à l'utilisation de l'outil `Tesseract` qui est un moteur entraîné avec des textes imprimés. Cependant, il est possible d'entraîner `Tesseract` pour mieux reconnaître l'écriture manuscrite. En général, les universités qui ont tenté cette approche ont obtenu des résultats de précision dépassant environ 90% [6].

4.2 Un point sur l'organisation

4.2.1 *Déroulement général*

Le stage s'est déroulé sur une durée de 11 semaines approximativement. Durant ce temps, j'ai pu beaucoup avancer dans la réalisation du projet. Le stage a commencé par une formation sur les outils essentiels au développement informatique ainsi qu'une explication sur le but du projet. J'ai ensuite commencé à concevoir une solution en se basant sur les informations données par le maître de stage avant de passer à la programmation

6. Pour plus de détails : https://tesseract-ocr.github.io/tessdoc

de la solution. La figure 5 représente la visualisation dans le temps des différentes étapes de la réalisation du projet pendant toute la durée du stage.

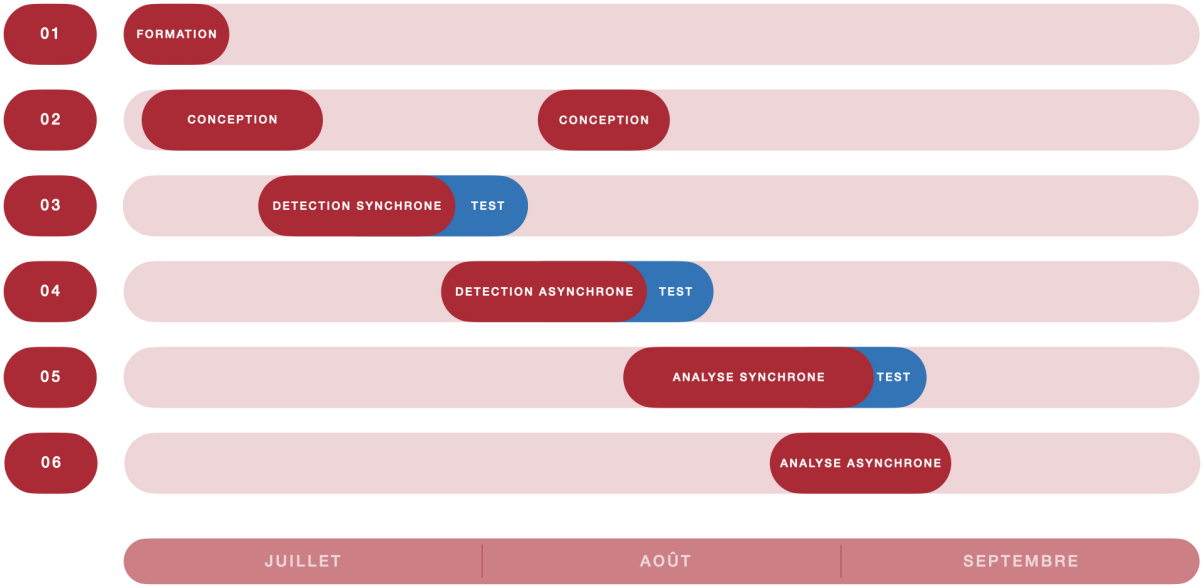

FIGURE 5 – Diagramme de Gantt de la réalisation

4.2.2 Principes de l'agilité

Bien que le stage n'ait pas été effectué en équipe, plusieurs principes de l'agilité ont été respectés, notamment les suivants :

- La concentration sur la qualité du développement logiciel.
- Les échanges quotidiens avec le maître du stage ainsi qu'avec les autres stagiaires.
- L'adaptation au changement tout en respectant le plan général du projet.

Références

[1] Hucker Marius : A table detection, cell recognition and text extraction algorithm to convert tables in images to excel files : `towardsdatascience.com`

[2] Wikipedia : OpenCV : `https://fr.wikipedia.org/wiki/OpenCV`

[3] Wikipedia : Tesseract (logiciel) : `https://fr.wikipedia.org/wiki/Tesseract_(logiciel)`

[4] caractéristiques et utilisation de Spring : https://www.baeldung.com/learn-spring-course
[5] Wikipedia : Spring : https://fr.wikipedia.org/wiki/Spring_(framework)

Annexes

A Structure des fichiers principaux du projet

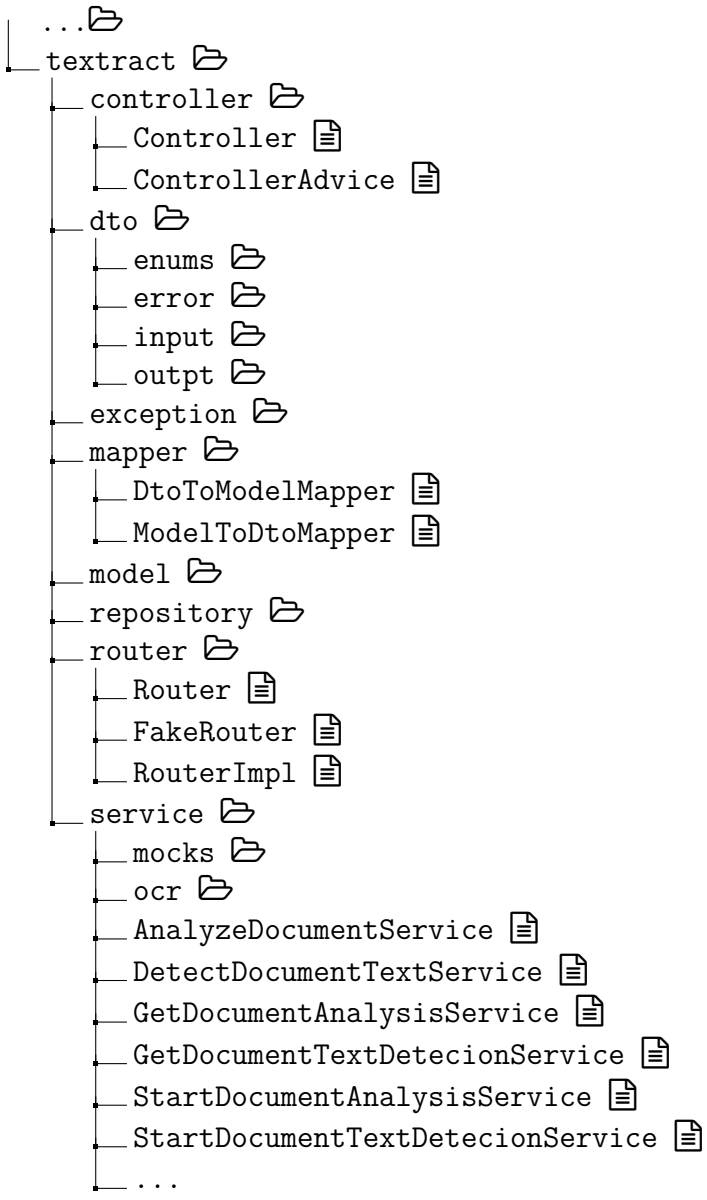

FIGURE 6 – Les fichiers principaux du projet