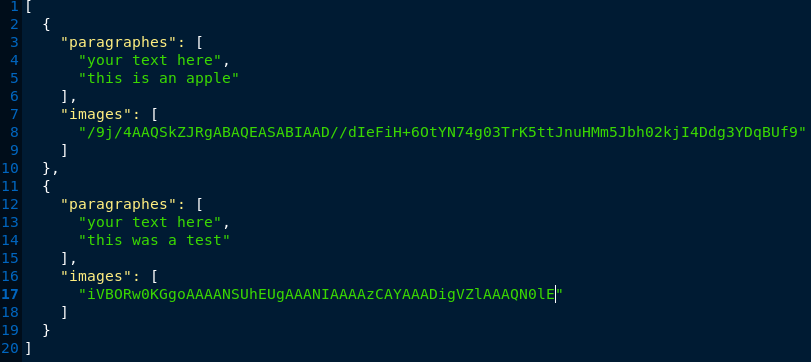
Rapport TP2

# Programme P1.js

Pour le programme P1 qui prend en entrée le texte et les images qui seront traitées par P2 et P3, nous avons choisi d’utiliser JavaScript comme langage.

Étant donné que nous pouvions choisir entre télécharger le texte et les images directement depuis Internet dans le programme ou depuis une source locale, nous avons choisi de se construire un format de fichier JSON qui contient les entrées avec le texte et les images correspondantes.

Voici à quoi ressemble notre format de fichier JSON :



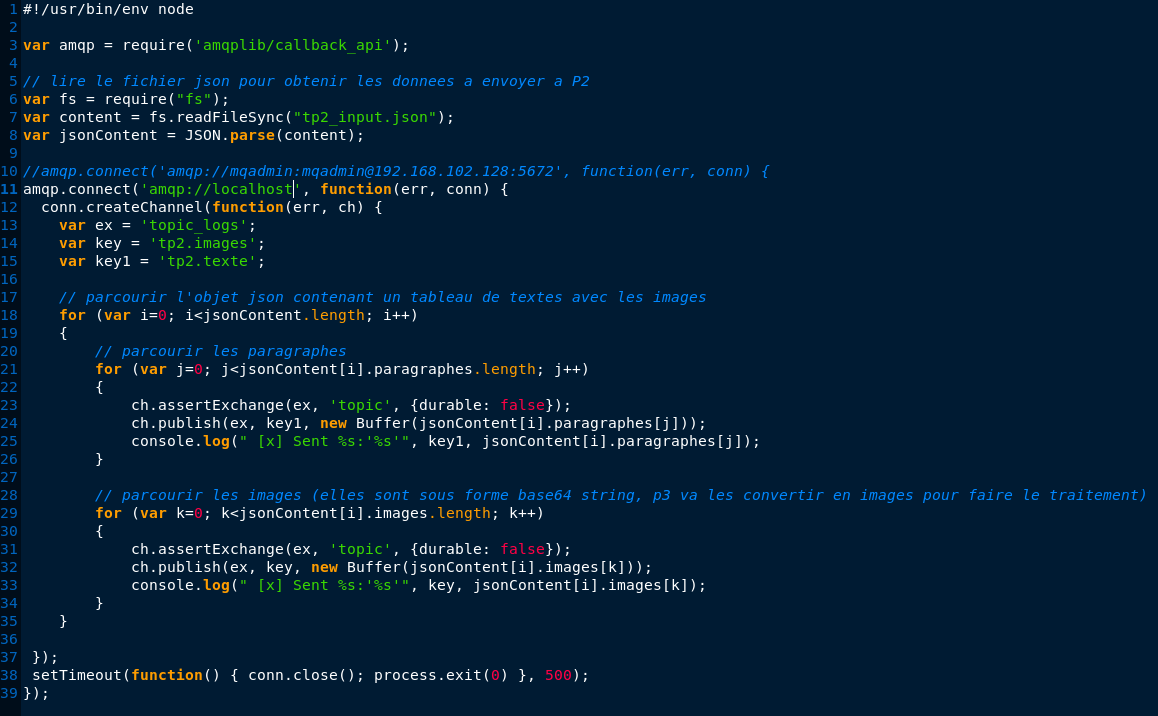
On peut voir que le contenu du fichier est principalement une liste d’objets qui contiennent eux deux listes, une liste de paragraphes et une liste d’images.

Les images sont encodées sous forme de chaîne de caractères en base64.

Le programme P1 est conçu pour lire tout le fichier et boucler sur tous les objets qui contiennent le texte et les images.

Ensuite, dans les deux boucles, une pour le texte et l’autre pour les images, on se connecte à un échangeur topic avec RabbitMQ, afin d’envoyer à la bonne file les images qui seront récupérées par le programme P3 écrit en java qui fera le redimensionnement et le texte qui sera récupéré par le programme P2 écrit en java aussi, qui fera la traduction du texte, tel que demandé.

Voici une capture du code de cette partie, la plus courte de notre application :



## Problèmes rencontrés

Les principaux problèmes rencontrés dans cette section sont liés à mon apprentissage de rabbitmq et de NodeJs, étant donné mon peu d’expérience avec ces technologies.

En effet, au départ, je ne savais pas comment lire mon fichier JSON avec NodeJs et comment parcourir la liste du texte et des images, j’ai donc dû commencer par me documenter avant de faire ma partie.

Ensuite, j’ai rencontré des problèmes en essayant de me connecter à la file de rabbitmq dans chacune des deux boucles, celle du texte et celle des images. Je n’arrivais pas à envoyer

....

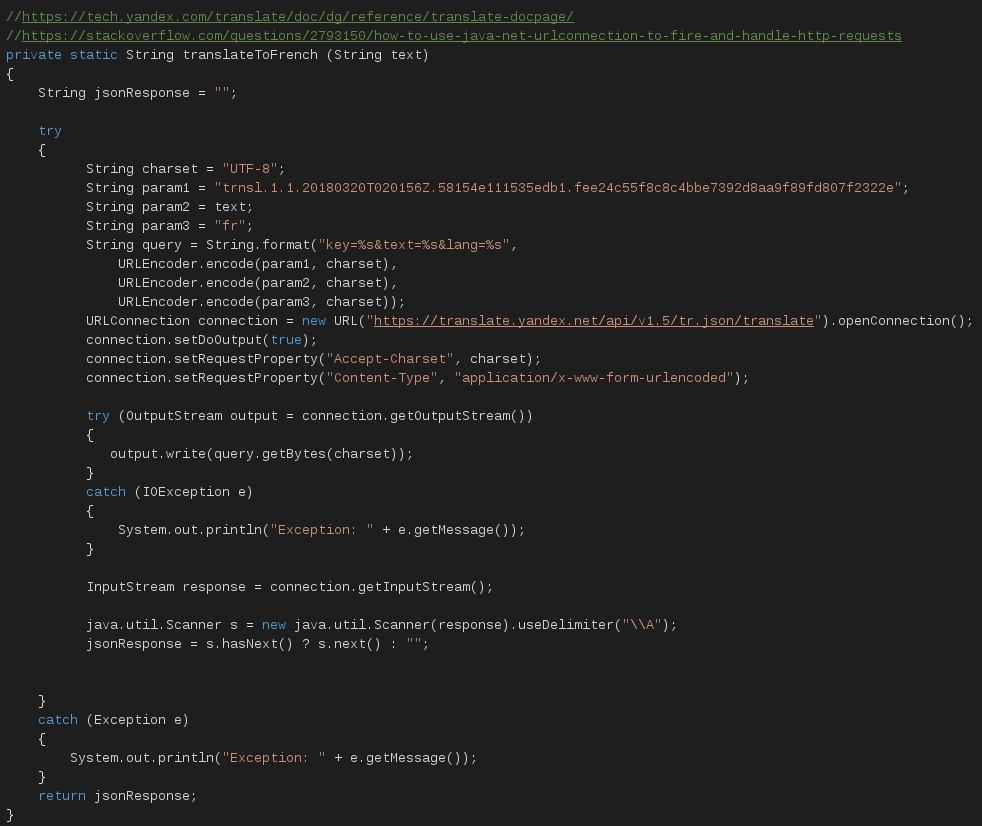
à compléter par alexandre

# Programme P2.java

Pour le programme P2, nous avons choisi Java comme langage.

Une fois la connexion à RabbitMQ effectuée et un canal de communication pour la réception des messages établi, la classe P2 se met écoute sur la file d’attente « tp2.texte ».

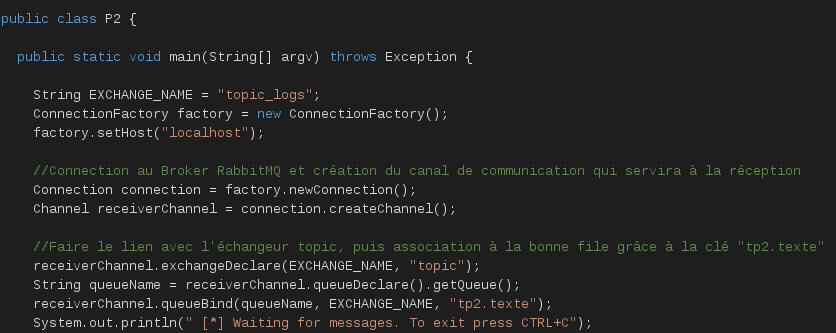
Lorsque du texte est reçu, celui-ci est envoyé à la méthode privée *translateToFrench*, dont voici une capture du code :

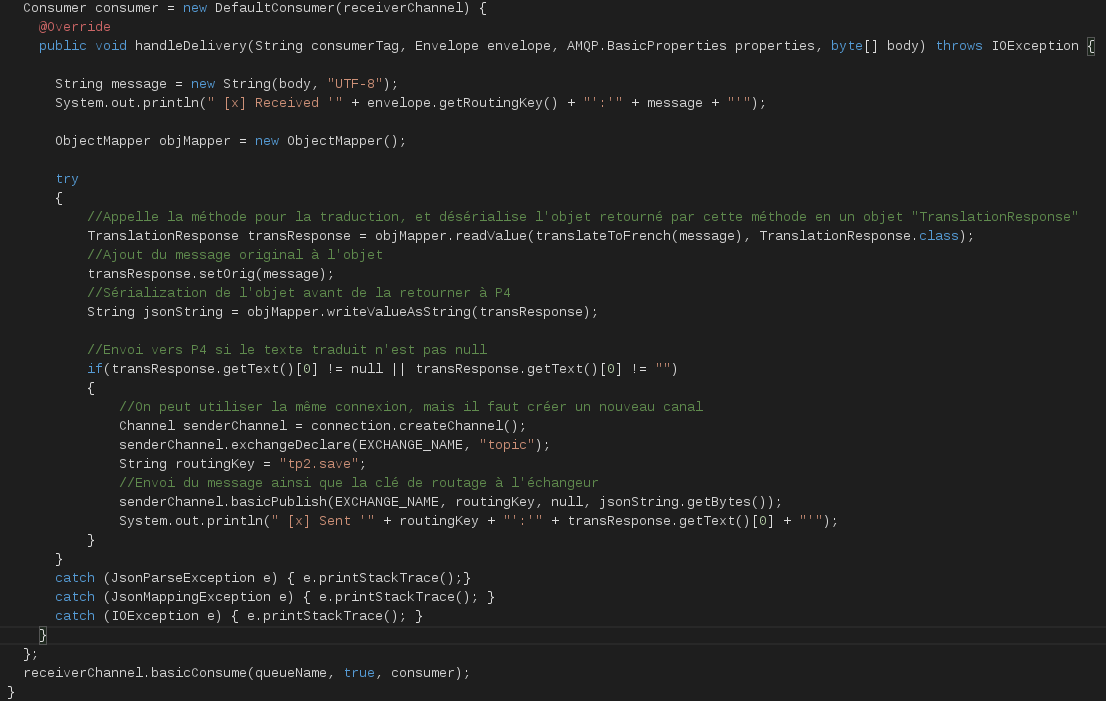


Cette méthode appelle une API gratuite du moteur de recherche Yandex qui s’occupe de faire la traduction du texte, que ce soit un seul mot ou un paragraphe entier.

La réponse en JSON reçue de cette API est ensuite désérialisée vers un objet de type TranslationResponse, puis on ajoute le texte original à cet objet. L’objet est ensuite resérialisé en JSON et prêt à l’envoi vers P4. La classe TranslationResponse est décrite plus loin dans ce document.

Pour l’envoi vers P4, un nouveau canal de communication est créé, puis on envoi le JSON créé précédemment vers l’échangeur avec la clé de routage « tp2.save ».

Voici un aperçu de la méthode *main*, séparée en deux captures d’écran : 



## Problèmes rencontrés

Un de mes problèmes avec le programme P2 a été de trouver un service gratuit qui offrait la traduction par une API. La plupart des solutions existantes sont payantes. Finalement, je suis tombé par hasard dur Yandex, qui correspondait exactement à nos besoins.

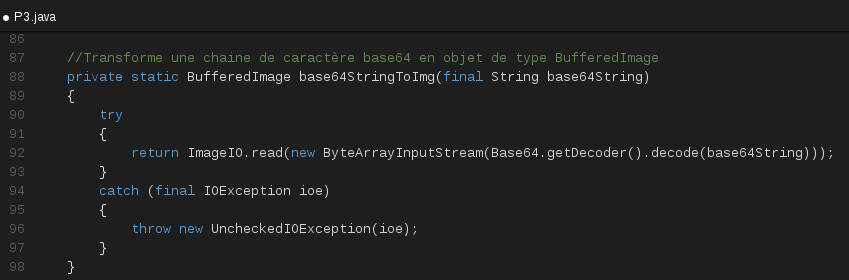
Le deuxième problème a été de savoir à quel moment ou à quel endroit dans le code je devais envoyer ma réponse vers P4 lorsque le texte était traduit. J’ai fini par comprendre que le code doit être placé directement dans la méthode « handleDelivery » afin d’être exécuter à la suite de la réception d’un message sur la file d’attente.

# Programme P3.java

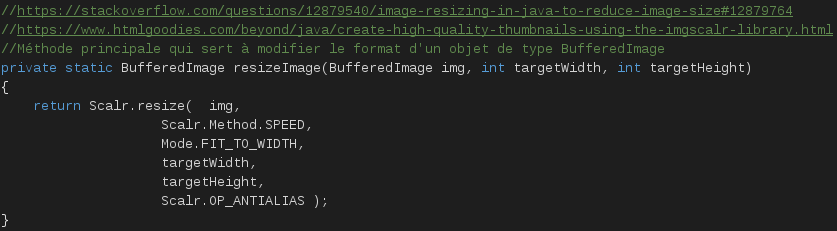
Pour le programme P2, nous avons choisi Java comme langage.

Une fois la connexion à RabbitMQ effectuée et un canal de communication pour la réception des messages établi, la classe P2 se met écoute sur la file d’attente « tp2.images ».

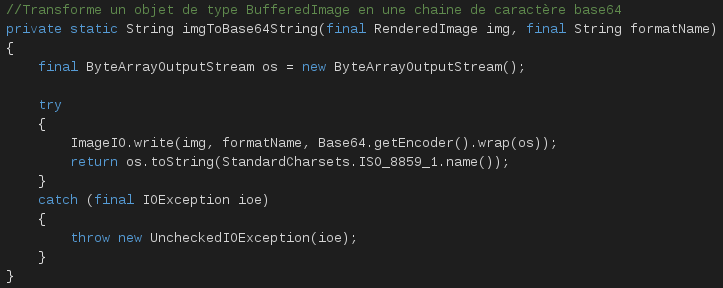
Lorsqu’une image sous format Base64 est reçue, cette image est convertie en objet de type BufferedImage à l’aide de la méthode privée *base64StringToImg*.



Une fois l’image dans ce format, la méthode privée *resizeImage* est appelée avec les paramètres de hauteur et de largeur voulus pour la transformation de l’image originale. Comme nous voulons une image rapetissée et une agrandie, elle est appelée deux fois.



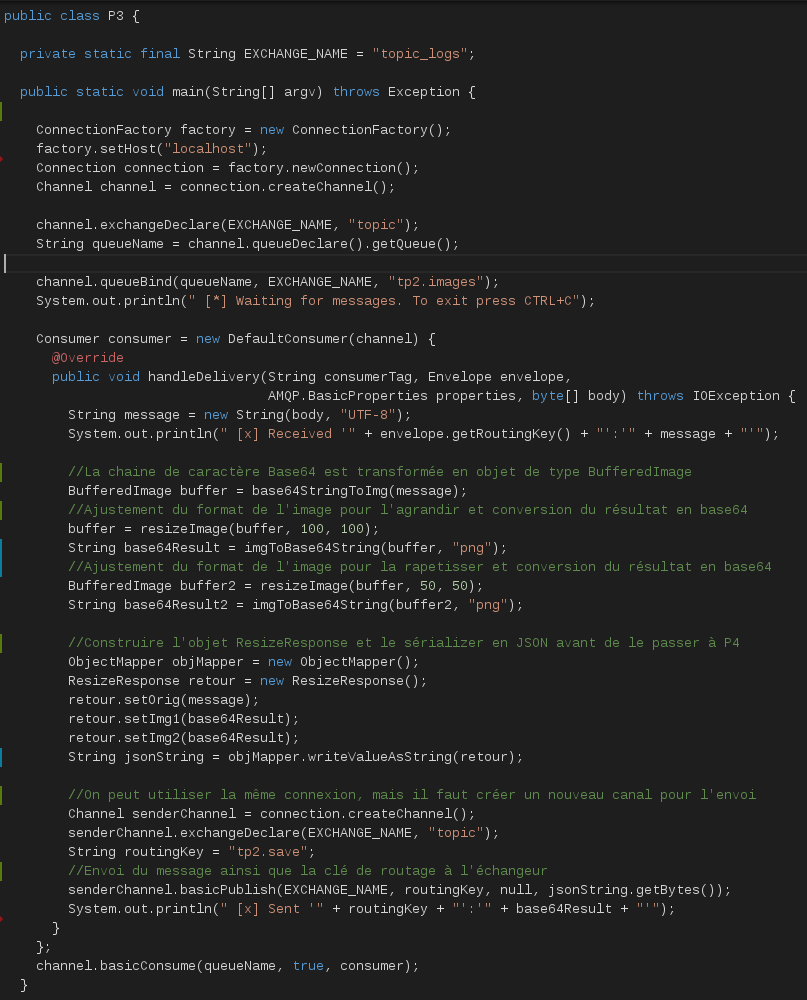
Les images obtenues sont ensuite reconverties en base64 grâce à la méthode privée *imgToBase64String*.



Nous avons maintenant trois chaines de caractère, correspondant chacune à une image de différentes tailles. Un objet de type ResizeResponse est alors construit avec ces chaines de caractère, puis cet objet est sérialisé en JSON avant d’être envoyé vers P4.

Pour l’envoi vers P4, un nouveau canal de communication est créé, puis on envoi le JSON créé précédemment vers l’échangeur avec la clé de routage « tp2.save ».

Voici une capture d’écran de la méthode *main* de P3 :



## Problèmes rencontrés

La modification de la taille des images m’a posé quelques problèmes, en commençant par trouver une librairie simple à utiliser qui offrait cette fonctionnalité. J’ai trouvé la librairie ImgScalr, mais très peu de documentation la concernant. Finalement son utilisation s’est révélée plutôt simple. Au niveau de RabbitMQ, j’ai eu peu de problèmes car j’avais déjà réalisé P3.

# Programme P4.java

À faire par Alexandre

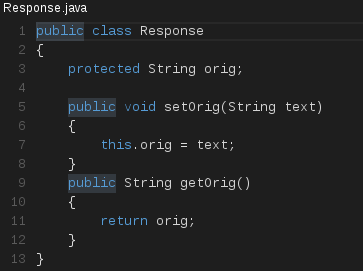
## Problèmes rencontrés

# DatabaseHelper.java

À compléter par Alexandre

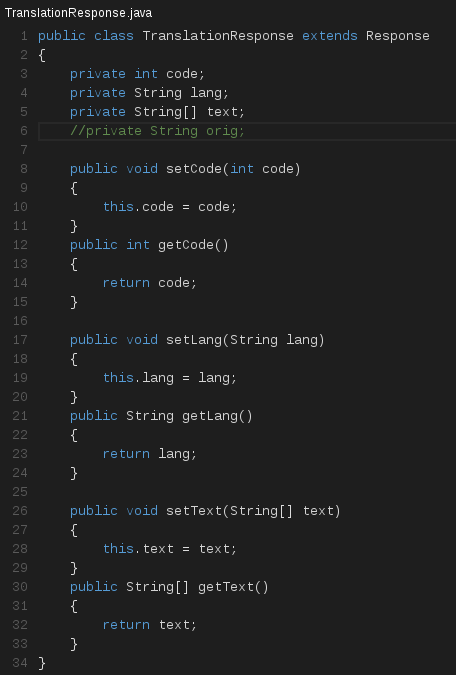
# Response.java

Classe mère des classes TranslationResponse et ResizeResponse, créée car ces deux classes contiennent la propriété « orig ». Effectivement, dans les deux cas l’image ou le texte original doit être réacheminé vers P4.



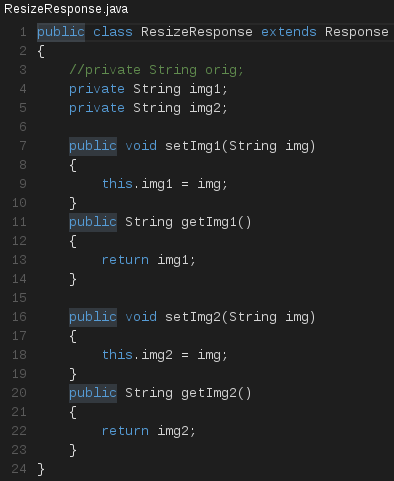
# TranslationResponse.java

Cette classe sert uniquement à la sérialisation/désérialisation JSON avec la libraire Jackson. Elle est utilisée dans la classe P2 et hérite de Response.java.

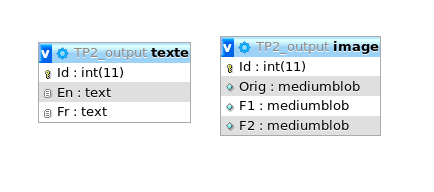


# ResizeResponse.java

Cette classe sert uniquement à la sérialisation/désérialisation JSON avec la libraire Jackson. Elle est utilisée dans la classe P3 et hérite de Response.java.



## Schéma de la base de données



## Script de création des tables

