TP N°5

Présentation

Dans ce TP nous allons mettre en œuvre le service AMQP via RabbitMQ pour réaliser du routage de données en fonction du destinataire. Pour cela récupérer à l'endroit habituel le correctif du TP N°4. Vous devez réutiliser le mécanisme « .env » pour le paramétrage de l'accès à RabbitMQ.

Architecture

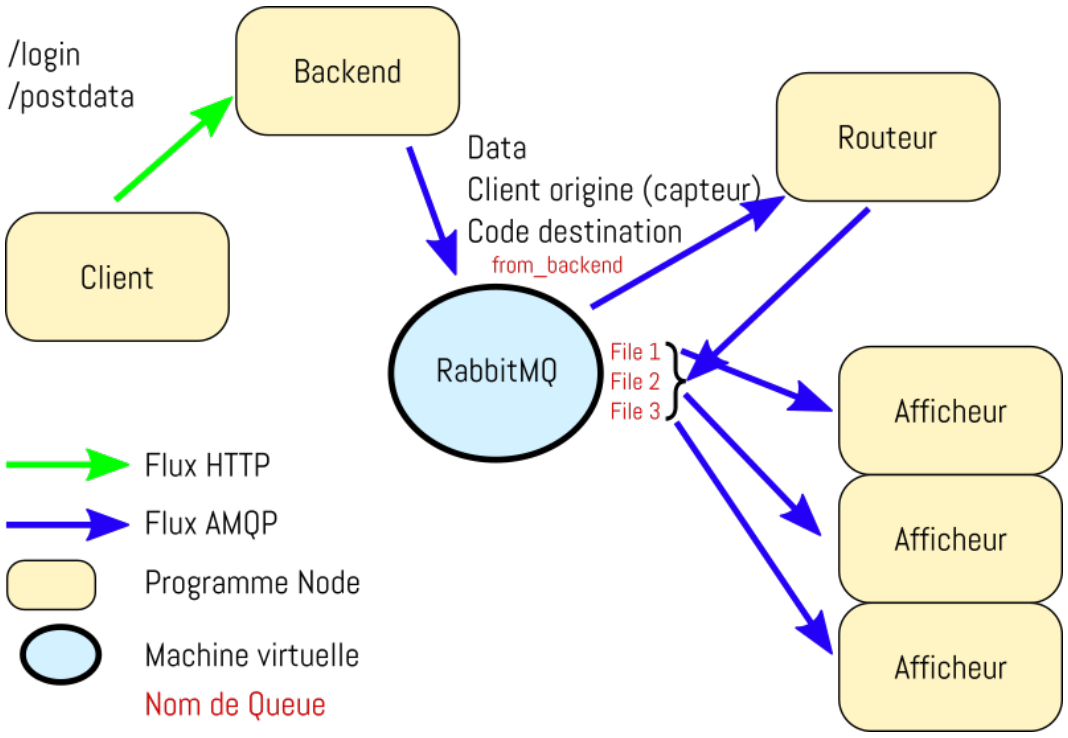

Figure 1:

Client

Vous pouvez repartir du client proposé en correction sous le nom client2.js. Celui-ci utilise le module node Axios(https://axios-http.com/docs/intro/) qui simplifie les accès à une API REST.

De même, vous pouvez supprimer le mécanisme de Pulling si vous le souhaitez.

Étape 1 :

Modifier le client de façon à transmettre un code de destinataire. Le code destinataire sera un entier qui activera correctement le routage de la donnée suivant le principe suivant :

Code	Destinataire
0	Interne (traitement chez nous)
1	Transmission externe (via un file d'attente N°1)
2	Transmission externe (via un file d'attente N°2)

Étape 2 :

Modifier le client de façon à accepter en paramètre ce code de destinataire. Utiliser le module npm « argparse » pour gérer les paramètres. Ainsi, le login sera passé par « -l » ou « –login », le mot de passe par « -p » ou « –password » et le destinataire par « -t » ou « –to ».

Backend

Notre serveur prendra maintenant le nom de backend. Il servira uniquement à recevoir les données sans aucun traitement. Il conservera, pour le moment, la fonction d'authentification.

Étape 1 :

Actuellement notre backend affiche les données qu'il reçoit. Modifier ce comportement pour qu'il dépose dans une file d'attente ***durable***. Nommer cette file d'attente « from_backend ».

Étape 2 :

Dans un fichier séparé qu'on nommera « authdb.js » créer un tableau d'identifiant/mot de passe pour 10 clients différents. Modifier le code du backend de façon à utiliser ce tableau pour identifier et authentifier les clients.

Routage

Nous allons maintenant créer un service de routage des données (le routeur).

Étape 1 :

Créer ce nouveau programme (le routeur) qui se chargera de récupérer les données depuis la file d'attente « from_backend » et qui, en fonction du code de destination, transmettra la donnée à la file d'attente appropriée. Créer pour cela un tableau qui associera le numéro de destinataire avec le nom de la file d'attente (déterminez les noms, ce sera les « files d'attente de destination »). Stocker ce tableau dans un fichier à part qu'on nommera « routingdb.js »

Étape 2 :

Modifier le fichier authdb.js de façon a inclure, pour chaque identifiant, la liste des codes de destinataire autorisé. Par exemple :

- Client 1 : 0 et 2
- Client 2 : uniquement 1
- Client 3 : 1, 2 et 3
- ...

Étape 3 :

Modifier le routeur de façon à contrôler la légitimité du trafic en comparant la destination demandée avec celles autorisées. Ignorer le trafic « illégal ».

Afficheur et finalisation

Nous allons créer un service d'« affichage » des données simple associé à chaque destination.

Étape 1 :

En se basant sur le programme « receiver.js » créer un afficheur pour une file d'attente de destination. La file d'attente à traiter sera récupérée en paramètre avec le module argparse et le paramètre « -t » ou « –to ».

Étape 2 :

Structurer le projet proprement :

- bin/ ne devrait contenir que les « programmes » exécutables. Le code écrit devrait être minimal
- src/ contiendra les différents code commun au projet (schéma, bibliothèques. . .)
- src/display/ devrait contenir le code principal de l'afficheur
- src/router/ devrait contenir le code principal du routeur
- src/client/ le code du client
- src/db les bases de données (authdb et routingdb)

Étape 3 :

Tester l'intégralité de la chaîne :

- Vérifier l'authentification
- Vérifier que le routage est correct (acceptations et rejets)
- Vérifier que l'affichage pour chaque destination dépile bien les messages autorisés