PLMS Assignment 4

PLMS - Building Microservices using NodeJS Assignments

Tasks:

1. Creating notification-service.
    a. Setting up the project.
    b. Connecting to MongoDB and RabbitMQ.
    c. Creating schemas for the routes.
    d. Creating the two routes.

Task 1. Creating notification-service

In this task, we will be working on a new service entirely for user notification. It will listen to both the queues together and then save them into the database.

1. Set up the auth-service

Step 1. Set up the service structure like all our previous services and this time we will do the same for `review-service`. Create a new folder called `review-service` and initialize a `package.json` file and install all these packages:

a) dotenv
b) express
c) jsonwebtoken
d) cors
e) mongoose
f) nodemon
g) zod
h) amqplib

Step 2. We have already created error classes and middleware; we can just copy them over to this new service as they created by keeping it reusable for other services too. Now we are going to set our basic express, cors. Set the PORT for this app to 8005. Remember to also go over the CORS as well if you are using the frontend.

2. Connecting to MongoDB and RabbitMQ

Step 1. Connect to MongoDB.

This time create a new cluster or an entire new MongoDB server and call it "`PLMS-Review`". Next we start working on the model. They should include the following:

- userid (string, required)
- loanId (string, required)
- role (enum:["customer", "manager"], required)
- type (enum:["loan_approved", "loan_rejected", "loan_created"], required)
- title (string, required)
- message (string, required)
- isRead (Boolean, required)
- createdAt (Date, required)

Now remember to also connect RabbitMQ to your app using the same setup as that from the previous services. Wire the listeners to the app so that the queue gets updated when a new message has been sent.

3. Creating schemas for the routes.

Now we are going to work on the schema for the routes. There are a total of 2 routes for this service, and they are as follows:

a) get-notifications-route – to fetch notifications of user.
b) mark-read-route – which marks the notification as read.

There would be no scheme for request to get-notifications-route because this route would directly use the JWT token to fetch the user's notifications.

As for the other route mark-read-route this will involve the marking the notification as read. Again, we do not need any kind of scheme for this route as we will be using the dynamic route to find the notification and then simply mark them as read after checking if the notification was marked by the right person.

4. Creating two routes.

Step 1. Creating get-loan-route.

Under `src/routes`, create `get-notifications-route.js`. Define `GET` request at ("`/api/notifications/all`").

Create a controller for this route where you need to fetch all notification data from the database. This will not be done using any data but rather the JWT Token that we will attach to the Authorization header of any request. This route we will have checks for current user role, and we proceed to fetch all notification for the user.

Step 2. Creating mark-read-route

Under `src/routes`, create `mark-read-route.js`. Define `POST` request at ("`/api/notifications/read/:id`").

Create a controller for this route where you need to mark the user's notification as read. Here we will first get user's id and role from the JWT token then divide the customers and manager separately. While all the managers get the same notifications, we will only show the customers' notifications to their respective customers, and we then send out the customer details.

Once you have successfully implemented everything then test it out on Postman and verify your results.

Congrats! You have successfully finished assignment 4. You are almost there! Have some good rest and see you tomorrow!