PLMS Assignment 3

PLMS - Building Microservices using NodeJS Assignments

Requirements:

1. RabbitMQ

Tasks:

1. Creating a Review service.
   a. Setting up the project.
   b. Connecting to MongoDB.
   c. Creating schemas for the routes.
   d. Creating the two routes.
2. Connect RabbitMQ to your services.
   a. Wire RabbitMQ to services.
   b. Wire queues listeners.

Task 1. Creating a Review Service

Now we'll create a new authentication service to handle user registration and login.

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

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 8004. Remember to also go over the CORS as well if you are using the frontend.

2. Connecting to MongoDB

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)
- updatedUserData (object with user details)
    a) name (String, required)
    b) dob (Date, required)
    c) profileURL (URL, required)
    d) gender (enum: ["male", "female", "prefer not to say"], required)
    e) employer (String, required)
    f) monthlySalary (Number, required)
    g) creditScore (Number, required)
- bankId, bankName (String, required)
- bankLogo (URL, required)
- appliedLoanAmount, interestRate, termLength (Number, required)
- dateOfApplication (default: current date)
- status (enum: ["pending", "approved", "rejected"], default: "pending")

3. Creating schema 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-loans-route – to fetch all user's loan data.
b) review-loan-route – to review a loan application created by the customer

There would be no scheme for request to get-loans-route. We are going to just use JWT to verify that the request is from a "`manager`" role and not customer trying to access this service.

The scheme for review-loan-route is going to include:

1) status (enum:["approved", "rejected"], required)

We are only using one single request data because we will be using dynamic route for reviewing of loan applications.

4. Creating the two routes.

Step 1. Creating get-loan-route.

Under `src/routes`, create `get-loans-route.js`. Define `GET` request at (“`/api/review/all`”).

Create a controller for this route where you need to fetch all loan 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 applications for the manager.

Step 2. Creating create-loan-route

Under `src/routes`, create `get-loan-route.js`. Define `GET` request at (“`/api/review/ update/:id`”).

Create a controller for this route where the dynamic `id` is used to find the loan. This id is going to be used for reference. We must also check if the user is manager or not before we proceed further. We can check if the loan is already reviewed or not with the status field in the loan data. Then, finally update the loan status to the status from the request body.

Task 3. Connect RabbitMQ to your services

In this task, we are going to be connecting our services using RabbitMQ and then we can procced with intercommunication.

1. Wiring RabbitMQ to services.

Step 1. Get your RabbitMQ service running (weather through cloud, local system, or docker). Then we need to install some new packages into all our services.

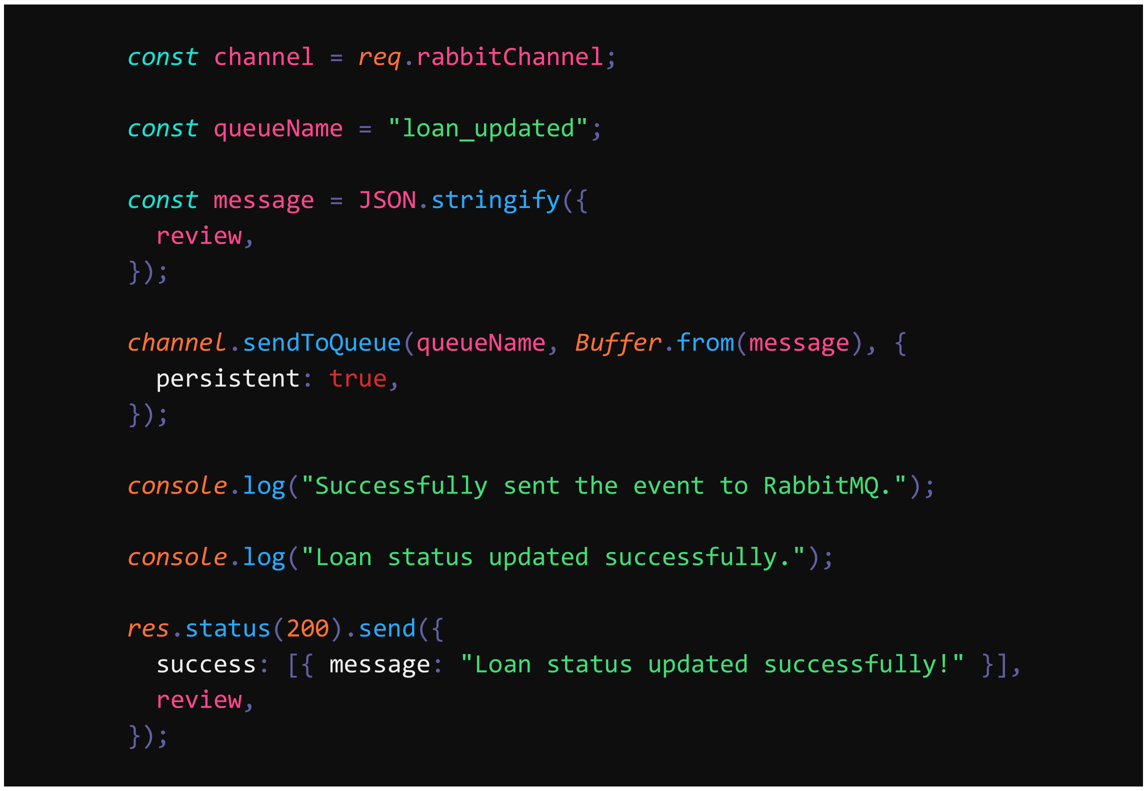
```
npm install amqplib
```

Once all the services have this package installed then we can start working on wiring the channel to our services. You can take this as example to connect to your RabbitMQ server.

```
amqp.connect(process.env.AMQP_CONNECT, (error0, connection) => {
  if (error0) {
    throw error0;
  }

  connection.createChannel((error1, channel) => {
    if (error1) {
      throw error1;
    }

    const app = express();

    app.use(express.json());

    app.use((req, res, next) => {
      req.rabbitChannel = channel;
      next();
    });


  // Our previous express app

 process.on("beforeExit", () => {
      console.log("Closing Connection to RabbitMQ!");
      connection.close();
    });
  });
});
```

<u>Note:</u> Remember to create a new environment variable for `AMQP_CONNECT`*. So that we can connect our service to RabbitMQ servers.*

Step 2. We can now make use of the rabbitChannel from the request variable. You can take this example of RabbitMQ for review of loans.



```
const channel = req.rabbitChannel;

const queueName = "loan_updated";

const message = JSON.stringify({
  review,
});

channel.sendToQueue(queueName, Buffer.from(message), {
  persistent: true,
});

console.log("Successfully sent the event to RabbitMQ.");

console.log("Loan status updated successfully.");

res.status(200).send({
  success: [{ message: "Loan status updated successfully!" }],
  review,
});
```

Using this same procedure create another queue for "`loan_updated`" in `review-loan-route` from our loan service.

2. Wire queue listeners.

Step 1. Now we need to add queue listeners, so we need to go back to our `index.js` file. Here, we need to create a listener right before we start the service. Here is an example for a "`loan_created`" queue.

```
channel.assertQueue("loan_created", { durable: true }, (error) => {
  if (error) {
    console.error("Queue assertion error:", error);
    return;
  }

  console.log("Asserted 'loan_created' queue");

  channel.consume("loan_created", saveLoans, { noAck: false });
});
```

Like any route handler, we also need to handle this queue request we receive from RabbitMQ. Let us create another controller (say "`updatedLoans`"). In this we first need to parse the data then check if the data is already updated or not. If not then we can update it using the controller.

Do the same for loan_created queues as well. A plus can be if you can also write another controller where the service tries to update itself but listen to the same queue it creates.

You can now test the intercommunication between services. If you are able to updated and create queues successfully, then congratulations on completing today's assignment.

Congrats~ You have reached the end of this assignment 3, and we're very close to the finishing line of this project! Have some good rest and see you tomorrow! :D