Develop Back-end Integration for Login

Estimated duration: 30 mins

Welcome to **Develop Backend Integration for Login**. In this lab, part of your capstone project, you'll develop an API endpoints for the `giftlink-backend` project. This exercise is designed to enhance your skills in Node.js and MongoDB by focusing on building and testing APIs routes.

Objectives

- Create an endpoint to login a user `/login` in `authRoutes.js`
- Configure the `api/auth` route in the Express application (`app.js`) to use the routes defined in authRoutes.js.
- Handle errors and structure responses in a server environment
- Push the implemented changes to GitHub

Prerequisite

- Ensure you have completed the previous modules and pushed all the changes to Github.
- You should have the updated `giftlink-backend` directory in your repository cloned to your local environment, or you will clone the repository that contains all of your pushed code. This will be your working directory for the lab

Step 1: Implement the /login Endpoint

In this step, you implement the `/login` endpoint in **authRoutes.js** file to act as middleware. It should create an authentication route to POST details using the login API route.

Tasks

1. Connect to `giftsdb` in MongoDB through `connectToDatabase` in `db.js`.
2. Access MongoDB `users` collection
3. Check for user credentials in database
4. Check if the user entered password matches the stored encrypyted password and send appropriate message on mismatch
5. Fetch user details from database
6. Create JWT authentication if passwords match with user._id as payload
7. Send appropriate message if user not found

Code Template

```
router.post('/login', async (req, res) => {
```

```
    try {
        // Task 1: Connect to `giftsdb` in MongoDB through `connectToDatabase` in `db.js`.
        // Task 2: Access MongoDB `users` collection
        // Task 3: Check for user credentials in database
        // Task 4: Task 4: Check if the password matches the encrypyted password and send appropriate message on mismatch
        // Task 5: Fetch user details from database
        // Task 6: Create JWT authentication if passwords match with user._id as payload
        res.json({authtoken, userName, userEmail });
        // Task 7: Send appropriate message if user not found
    } catch (e) {
         return res.status(500).send('Internal server error');
    }
});
```

Hints

Task 1: Connect to `giftsdb` in MongoDB through `connectToDatabase` in `db.js`.

▶ Click here for a hint

Task 2: Access the `users` collection

▶ Click here for a hint

Task 3: Check user credentials

▶ Click here for a hint

Task 4: Check if the password matches the encrypyted password and send appropriate message on mismatch

▶ Click here for a hint

Task 5: Fetch user details in database

▶ Click here for a hint

Task 6: Create JWT authentication if passwords match with user._id as payload

▶ Click here for a hint

Task 7: Send appropriate message if user not found

▶ Click here for a hint

Step 2: Push changes to GitHub

After implementing and testing the API endpoints, configure your git username and email through the terminal and push your changes to GitHub.

Tasks

Task 1: Stage changes
Task 2: Commit changes
Task 3: Push changes

Hints

1. Hint for staging changes

▶ Click here for a hint

2. Hint for committing changes

▶ Click here for a hint

3. Hint for pushing to GitHub

▶ Click here for a hint

Step 3: Evidence and Final Solution

Your final solution should look similar to the file provided below.

Final Solution

Solution for `routes/authRoutes.js` including both `/register` and `/login` APIs

▼ Click here for a hint

```
const express = require('express');
const bcryptjs = require('bcryptjs');
const jwt = require('jsonwebtoken');
const connectToDatabase = require('../models/db');
const router = express.Router();
const dotenv = require('dotenv');
const pino = require('pino');  // Import Pino logger
dotenv.config();
const logger = pino();  // Create a Pino logger instance
```

```
//Create JWT secret
dotenv.config();
const JWT_SECRET = process.env.JWT_SECRET;
router.post('/register', async (req, res) => {
    try {
      //Connect to `giftsdb` in MongoDB through `connectToDatabase` in `db.js`.
      const db = await connectToDatabase();
      //Access the `users` collection
      const collection = db.collection("users");
      //Check for existing email in DB
      const existingEmail = await collection.findOne({ email: req.body.email });
        if (existingEmail) {
            logger.error('Email id already exists');
            return res.status(400).json({ error: 'Email id already exists' });
        }
        const salt = await bcryptjs.genSalt(10);
        const hash = await bcryptjs.hash(req.body.password, salt);
        const email=req.body.email;
        //Save user details
        const newUser = await collection.insertOne({
            email: req.body.email,
            firstName: req.body.firstName,
            lastName: req.body.lastName,
            password: hash,
            createdAt: new Date(),
        });
        const payload = {
            user: {
                id: newUser.insertedId,
            },
        };
        //Create JWT
        const authtoken = jwt.sign(payload, JWT_SECRET);
        logger.info('User registered successfully');
        res.json({ authtoken,email });
    } catch (e) {
        logger.error(e);
        return res.status(500).send('Internal server error');
    }
});
    //Login Endpoint
router.post('/login', async (req, res) => {
    console.log("\n\n Inside login")
    try {
        // const collection = await connectToDatabase();
        const db = await connectToDatabase();
        const collection = db.collection("users");
        const theUser = await collection.findOne({ email: req.body.email });
        if (theUser) {
            let result = await bcryptjs.compare(req.body.password, theUser.password)
            if(!result) {
                logger.error('Passwords do not match');
                return res.status(404).json({ error: 'Wrong pasword' });
            }
            let payload = {
                user: {
```

```
                    id: theUser._id.toString(),
                },
            };
            const userName = theUser.firstName;
            const userEmail = theUser.email;
            const authtoken = jwt.sign(payload, JWT_SECRET);
            logger.info('User logged in successfully');
            return res.status(200).json({ authtoken, userName, userEmail });
        } else {
            logger.error('User not found');
            return res.status(404).json({ error: 'User not found' });
        }
    } catch (e) {
        logger.error(e);
        return res.status(500).json({ error: 'Internal server error', details: e.message });
      }
});
module.exports = router;
```

Evidence

Make note of the GitHub URL for your repository. You will be asked to submit this for peer evaluation. Ensure you completed all tasks in each step.

You may use, `curl` command or Postman to ensure that your backend is working as expected.