

Develop Back-end Integration for User Profile

Estimated duration: 30 mins



Welcome to the **Develop the Back-end Integration for User Profile** lab. In this lab you'll develop an API endpoint 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 update user credentials in the database (/update) 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

Prerequisites

- 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 /update Endpoint

Add the code to the `/update` endpoint in `authRoutes.js` file so it can act as middleware. This should create an authentication route to update the profile. You can find the `authRoutes.js` file in the `routes` folder.

Tasks

1. Use the `body, validationResult` from `express-validator` for input validation
2. Validate the input using `validationResult` and return appropriate message if there is an error.
3. Check if `email` is present in the header and throw an appropriate error message if not present.
4. Connect to `giftsdb` in MongoDB through `connectToDatabase` in `db.js` and access `users` collection.
5. Find user credentials in database
6. Update user credentials in database
7. Create JWT authentication with `user._id` as payload using secret key from `.env` file

Code Template

```
// {Insert it along with other imports} Task 1: Use the `body`,`validationResult` from `express-validator` for input validation
router.put('/update', async (req, res) => {
  // Task 2: Validate the input using `validationResult` and return appropiate message if there is an error.
  try {
    // Task 3: Check if `email` is present in the header and throw an appropriate error message if not present.
    // Task 4: Connect to MongoDB
    // Task 5: find user credentials in database
    existingUser.updatedAt = new Date();
    // Task 6: update user credentials in database
    // Task 7: create JWT authentication using secret key from .env file
    res.json({authtoken});
  } catch (e) {
    return res.status(500).send('Internal server error');
  }
});
```

Hints

Task 1: Use the body,validationResult from express-validator for input validation

▼ Click here for a hint

```
const { body, validationResult } = require('express-validator');
```

Task 2: Validate the input using validationResult and return appropiate message if there is an error.

▼ Click here for a hint

```
const errors = validationResult(req);
if (!errors.isEmpty()) {
  logger.error('Validation errors in update request', errors.array());
  return res.status(400).json({ errors: errors.array() });
}
```

```
}
```

Task 3: Check if email is present in the header and throw an appropriate error message if not present.

▼ [Click here for a hint](#)

```
const email = req.headers.email;
if (!email) {
  logger.error('Email not found in the request headers');
  return res.status(400).json({ error: "Email not found in the request headers" });
}
```

Task 4: Connect to giftddb in MongoDB through connectToDatabase in db.js and access users collection.

▼ [Click here for a hint](#)

```
const db = await connectToDatabase();
const collection = db.collection("users");
```

Task 5: Access the user details

▼ [Click here for a hint](#)

```
const existingUser = await collection.findOne({ email });
```

Task 6: Update user credentials

▼ [Click here for a hint](#)

```
const updatedUser = await collection.findOneAndUpdate(  
  { email },  
  { $set: existingUser },  
  { returnDocument: 'after' }  
);
```

Task 7: Create JWT authentication with user._id as payload using secret key from .env file

▼ [Click here for a hint](#)

```
const payload = {  
  user: {  
    id: updatedUser._id.toString(),  
  },  
};  
const authToken = jwt.sign(payload, JWT_SECRET);
```

Step 2: Push changes to GitHub

Tasks

Task 1: Stage changes

Task 2: Commit changes

Task 3: Push changes

Hints for Git Commands

1. Hint for staging changes

▼ Click here for a hint

Use `git add .` to stage all your changes for commit

2. Hint for committing changes

▼ Click here for a hint

Commit your staged changes with `git commit -m "Your message here"`

3. Hint for pushing to GitHub

▼ Click here for a hint

Finally, push your commits to GitHub with `git push`

Step 3: Evidence and Final Solution

Solution

▼ Your `authRoutes.js` file should look similar to the code provided below.

```
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
//Task 1: Use the `body`, `validationResult` from `express-validator` for input validation
const { body, validationResult } = require('express-validator');
const logger = pino(); // Create a Pino logger instance
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();
    const collection = db.collection("users");
    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;
    console.log('email is',email);
    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,
      },
    };
    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');
  }
});
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 });
}
});
// update API
router.put('/update', async (req, res) => {
    // Task 2: Validate the input using `validationResult` and return appropriate message if there is an error.
    const errors = validationResult(req);
    // Task 3: Check if `email` is present in the header and throw an appropriate error message if not present.
    if (!errors.isEmpty()) {
        logger.error('Validation errors in update request', errors.array());
        return res.status(400).json({ errors: errors.array() });
    }
    try {
        const email = req.headers.email;
        if (!email) {
            logger.error('Email not found in the request headers');
            return res.status(400).json({ error: "Email not found in the request headers" });
        }
        //Task 4: Connect to MongoDB
        const db = await connectToDatabase();
        const collection = db.collection("users");
        //Task 5: Find user credentials
        const existingUser = await collection.findOne({ email });
        if (!existingUser) {
            logger.error('User not found');
            return res.status(404).json({ error: "User not found" });
        }
        existingUser.firstName = req.body.name;
        existingUser.updatedAt = new Date();
        //Task 6: Update user credentials in DB
        const updatedUser = await collection.findOneAndUpdate(
            { email },
            { $set: existingUser },
            { returnDocument: 'after' }
        );
        //Task 7: Create JWT authentication with user._id as payload using secret key from .env file
        const payload = {
            user: {
                id: updatedUser._id.toString(),
            },
        };
        const authToken = jwt.sign(payload, JWT_SECRET);
        logger.info('User updated successfully');
        res.json({ authToken });
    } catch (error) {
        logger.error(error);
        return res.status(500).send("Internal Server Error");
    }
});
});

```

```
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.

© IBM Corporation. All rights reserved.