npm install express jsonwebtoken bcrypt dotenv

**App.js**

//For JWT

const dotenv = require("dotenv");

dotenv.config();

let express = require("express");

let bodyParser = require("body-parser"); //to parse the incoming request bodies

/// const session = require('express-session'); //to store the session data on the server side

let connection = require("./dbcon.js");

let router = require("./routers.js");

let app = express(); //creates an Express application

app.use(bodyParser.json()); //to parse the incoming request bodies

app.use(bodyParser.urlencoded({ extended: true })); //to parse the incoming request bodies

app.use("/proj", router); //to use the router

app.listen(3000, function () {

  console.log("Server is running"); //to start a UNIX socket and listen for connections on the given path

});

**Routers.js**

require("dotenv").config();

let express = require("express"); //to create an Express application

let router = express.Router(); //to create a new router object

let connection = require("./dbcon.js"); //to create a connection with the database

let query = require("./queries.js"); //to write the queries

// For JWT

const bcrypt = require("bcrypt"); //to hash the password

const jwt = require("jsonwebtoken"); //to create a JSON Web Token

// const dotenv = require('dotenv');

// dotenv.config();

router.post("/register/", function (req, res) {

  const { U\_NAME, EMAIL, ROLE, PASSWORD } = req.body; //recieve the data from the request body

  if (U\_NAME == null || EMAIL == null || ROLE == null || PASSWORD == null) {

    return res.status(400).json({ message: "Please enter all the fields" }); //400: Bad request, server can't understand

  }

  const hashedPassword = bcrypt.hashSync(PASSWORD, 10); //hash the password, 10 is the saltRounds

  const values = [U\_NAME, EMAIL, ROLE, hashedPassword]; //values to be inserted in the database

  connection.query(query.registerUserQuery, [values], (error, result) => {

    //query to be executed

    if (error) {

      if (error.code === "ER\_DUP\_ENTRY") {

        return res.status(409).json({ message: "User Already Exists" }); //409: Conflict, user already exists

      }

      console.error("Error while registering the user", error);

      res.status(500).json({ message: "Error while registering the user" }); //500: Internal Server Error

    }

    res.send("User Registered Successfully");

  });

});

//for login

router.post("/login/", function (req, res) {

  const { EMAIL, PASSWORD } = req.body;

  if (EMAIL == null || PASSWORD == null) {

    res.status(400).json({ message: "Please enter all the fields" });

  }

  connection.query(query.loginUserQuery, [EMAIL], (error, result) => {

    if (error) {

      console.log("Error while logging in", error);

      return res.status(500).json({ message: "Error while logging in" });

    }

    if (result.length === 0) {

      console.log("User does not exist");

      return res.status(401).json({ message: "User does not exist" }); //401: Unauthorized, user does not exist

    }

    const user = result[0]; //as email is unique, only one user will be returned

    //compare the password entered with the hashed password in the database

    bcrypt.compare(PASSWORD, user.PASSWORD, (err, isMatch) => {

      if (err) {

        console.log("Error while comparing the passwords", err);

        res

          .status(500)

          .json({ message: "Error while comparing the passwords" });

      }

      if (!isMatch) {

        return res.status(401).json({

          message: "Invalid Credentials or Hashed Password not found",

        });

      }

      //payload is the data that is stored inside the token

      const tokenPayload = {

        email: EMAIL,

        role: user.ROLE,

        exp: Math.floor(Date.now() / 1000) + 60 \* 10, //it will expire after 10 minutes

      };

      const token = jwt.sign(tokenPayload, `${process.env.JWT\_SECRET}`); //secret key is used to sign the token

      //jwt.sign function takes the payload and the secret, and it returns a JWT as a string.

      return res.json({ message: "Login successful", token: token });

    });

  });

});

router.post("/insertBook/", authenticateToken, function (req, res) {

  const { BNAME, AUTHORS, GENRE } = req.body;

  // const userRole = req.body.role; // Get the user role from the request body

  const userRole = req.user.role;

  if (userRole !== "admin") {

    return res.status(403).json({ message: "Sorry, you are not an admin" });

  }

  if (!BNAME || !AUTHORS || !GENRE) {

    return res.status(400).json({ message: "Please enter all the fields" });

  }

  const values = [BNAME, AUTHORS, GENRE];

  connection.query(query.insertBookQuery, [values], (error, result) => {

    if (error) {

      if (error.code === "ER\_DUP\_ENTRY") {

        return res.status(409).json({ message: "Book Already Exists" });

      }

      console.error("Error while registering the user", error);

      return res

        .status(500)

        .json({ message: "Error while inserting the book" });

    }

    return res.status(201).json({ message: "Book inserted successfully" });

  });

});

router.delete("/delBook/", authenticateToken, function (req, res) {

  const { BID, BNAME, AUTHORS, GENRE } = req.body;

  // const userRole = req.body.role; // Get the user role from the request body

  const userRole = req.user.role;

  if (userRole !== "admin") {

    return res.status(403).json({ message: "Sorry, you are not an admin" });

  }

  if (!BID && !BNAME && !AUTHORS && !GENRE) {

    return res.status(400).json({ message: "Please enter at least one field" });

  }

  connection.query(

    query.delBookQuery,

    [BID, BNAME, AUTHORS, GENRE],

    (error, result) => {

      if (error) {

        console.error("Error while deleting the book", error);

        return res

          .status(500)

          .json({ message: "Error while deleting the book" });

      }

      return res.json({ message: "Book deleted successfully" });

    }

  );

});

router.put("/updateBook/", authenticateToken, function (req, res) {

  const { BID, BNAME, AUTHORS, GENRE } = req.body;

  // const userRole = req.body.role; // Get the user role from the request body

  const userRole = req.user.role;

  if (userRole !== "admin") {

    return res.status(403).json({ message: "Sorry, you are not an admin" });

  }

  if (!BID || !BNAME || !AUTHORS || !GENRE) {

    return res.status(400).json({ message: "Please enter all the fields" });

  }

  connection.query(

    query.updateBookQuery,

    [BNAME, AUTHORS, GENRE, BID],

    (error, result) => {

      if (error) {

        console.error("Error while updating the book", error);

        return res

          .status(500)

          .json({ message: "Error while updating the book" });

      }

      return res.json({ message: "Book updated successfully" });

    }

  );

});

router.get("/searchBook/", function (req, res) {

  const { BNAME, AUTHORS, GENRE } = req.body;

  if (!BNAME && !AUTHORS && !GENRE) {

    return res.status(400).json({ message: "Please enter at least one field" });

  }

  connection.query(

    query.getBooksQuery,

    [BNAME, AUTHORS, GENRE],

    (error, result) => {

      if (error) {

        console.error("Error while searching the book", error);

        return res

          .status(500)

          .json({ message: "Error while searching the book" });

      }

      return res.json(result);

    }

  );

});

function authenticateToken(req, res, next) {

  const authHeader = req.headers["authorization"]; // is used to extract JSON Web Token from the Authorization header in an HTTP request.

  const token = authHeader && authHeader.split(" ")[1];

  // Checks if the token is present in the Authorization header, if not then it will short circuit and return undefined

  if (!token) {

    return res.status(401).json({ message: "Access token not provided" });

  }

  jwt.verify(token, `${process.env.JWT\_SECRET}`, (err, user) => {

    //jwt.verify function takes the token and the secret, and it returns the payload if the signature is valid.

    if (err) {

      console.error("Error while verifying token", err);

      return res.status(403).json({ message: "Invalid token" });

    }

    req.user = user; //set the user in the request object

    next();

  });

}

module.exports = router;

**Queries.js**

const registerUserQuery =

  "INSERT INTO USERS (U\_NAME,EMAIL,ROLE,PASSWORD) VALUES (?)";

const loginUserQuery = "SELECT \* FROM USERS WHERE EMAIL = ?";

const insertBookQuery = "INSERT INTO BOOKS (BNAME, AUTHORS, GENRE) VALUES (?)";

const getBooksQuery =

  "SELECT \* FROM BOOKS WHERE BNAME = ? OR AUTHORS = ? OR GENRE = ?";

const delBookQuery =

  "DELETE FROM BOOKS WHERE BID = ? OR BNAME = ? OR AUTHORS = ? OR GENRE = ?";

const updateBookQuery =

  "UPDATE BOOKS SET BNAME = ?, AUTHORS = ?, GENRE = ? WHERE BID = ?";

module.exports = {

  registerUserQuery,

  loginUserQuery,

  insertBookQuery,

  getBooksQuery,

  delBookQuery,

  updateBookQuery,

};

**dbcon.js**

require('dotenv').config();

let mysql = require('mysql');

const connection = mysql.createConnection({

    host : 'localhost',

    user : 'root',

    password : 'password',

    database : 'bookdirectory'

});

connection.connect(error =>{

    if(error){

        console.log("There is some error while creating database connection", error);

        process.exit(1);

    }

    console.log("Database connected successfully")

})

module.exports = connection;

**dotenv.env**

JWT\_SECRET=Sheraz;

HOST=localhost;

USER=root;

PASSWORD=password;

DATABASE=bookdirectory;

**Tables:**



