Advanced Database Topics (COMP-8157)



**Assignment -1**

**Name : Karan Mahajan**

**Section : 3**

**Student ID : 110119774**

**DB Schema**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**userSchema.js**

**A screen shot of a computer program

Description automatically generated**

**blogSchema.js**

**A screen shot of a computer program

Description automatically generated**

**commentSchema.js**

**A screen shot of a computer screen

Description automatically generated**

**blog.js**

const express = require('express');

const router = express.Router();

const Blog = require('../model/blogSchema');

const User = require('../model/userSchema');

const Comment = require('../model/commentSchema');

// GET all blogs with optional filtering by tags and categories

router.get('/all', async (req, res) => {

try {

const { tags, categories } = req.query;

const filter = {};

if (tags) {

filter.tags = { $in: tags.split(',') };

}

if (categories) {

filter.categories = { $in: categories.split(',') };

}

const blogs = await Blog.find(filter);

res.status(200).json(blogs);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to fetch the blogs' });

}

});

// POST a new blog

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

try {

const { title, content, categories, tags } = req.body;

const { user } = req.userData;

const userDetails = await User.findOne({ email: user.email }).exec();

const blogData = {

title,

content,

categories,

tags,

authorName: `${userDetails.firstName} ${userDetails.lastName}`,

author: userDetails.id,

};

const userBlog = await Blog.create(blogData);

res.status(200).json(userBlog);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to upload the blog' });

}

});

// DELETE a blog and its associated comments

router.delete('/delete', async (req, res) => {

try {

const id = req.query.id;

await Blog.deleteOne({ \_id: id }).exec();

await Comment.deleteMany({ blogID: id }).exec();

res.status(200).json({ message: 'Successfully deleted' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to delete the blog' });

}

});

// PUT (update) a blog

router.put('/update/:blogId', async (req, res) => {

try {

const { title, content, categories, tags } = req.body;

const { blogId } = req.params;

const { user } = req.userData;

const userDetails = await User.findOne({ email: user.email }).exec();

if (!userDetails) {

return res.status(400).json({ message: 'User not found' });

}

const updatedData = {

title,

content,

categories,

tags,

authorName: `${userDetails.firstName} ${userDetails.lastName}`,

author: userDetails.id,

};

const updatedBlog = await Blog.findByIdAndUpdate(blogId, updatedData, { new: true }).exec();

if (!updatedBlog) {

return res.status(400).json({ message: 'Blog not found' });

}

res.status(200).json(updatedBlog);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to update the blog' });

}

});

// POST a comment on a blog

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

try {

const { comment, user, blogId } = req.body;

const userDetails = await User.findOne({ email: user }).exec();

const commentData = {

text: comment,

blogID: blogId,

commenterName: `${userDetails.firstName} ${userDetails.lastName}`,

};

await Comment.create(commentData);

res.status(200).json({ message: 'Successful' });

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to add the comment' });

}

});

// GET comments for a specific blog

router.get('/getcomments', async (req, res) => {

try {

const { id } = req.query;

const commentsData = await Comment.find({ blogID: id });

res.status(200).json(commentsData);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Unable to fetch the comments' });

}

});

module.exports = router;

**user.js**

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const User = require('../model/userSchema');

const saltRounds = 10;

const ValidateEmail = (mail) => {

// Use a regular expression to validate email.

return /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(mail);

};

const hashPassword = async (password) => {

try {

const hash = await bcrypt.hash(password, saltRounds);

return hash;

} catch (error) {

console.error(error.message);

throw error; // Rethrow the error for handling in the route handler.

}

};

const compareHashPassword = async (password, hashPassword) => {

try {

const isValid = await bcrypt.compare(password, hashPassword);

return isValid;

} catch (error) {

console.error(error.message);

throw error; // Rethrow the error for handling in the route handler.

}

};

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

try {

const { password, confirmPassword, firstName, lastName, email, mobile, DOB } = req.body;

if (!email || !ValidateEmail(email)) {

return res.status(400).json({ message: 'Invalid Email Address' });

}

if (!password || !confirmPassword || password !== confirmPassword) {

return res.status(400).json({ message: 'Passwords do not match' });

}

if (!mobile || !Number.isInteger(mobile)) {

return res.status(400).json({ message: 'Invalid Mobile Number' });

}

if (!firstName || !lastName) {

return res.status(400).json({ message: 'First Name or Last Name missing' });

}

console.log(DOB);

if (new Date(DOB) == 'Invalid Date') {

return res.status(400).json({ message: 'Invalid Date of Birth' });

}

const existingUser = await User.findOne({ email }).exec();

if (existingUser) {

return res.status(409).json({ message: 'User Already Exists' });

}

const newPassword = await hashPassword(password);

const newUserDetails = {

firstName,

lastName,

mobile,

password: newPassword,

email,

dob: DOB,

};

const user = await User.create(newUserDetails);

res.status(200).json(user);

} catch (error) {

console.error(error.message);

res.status(500).json({ message: 'Unable to register' });

}

});

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

try {

const { password, email } = req.body;

if (!email) {

return res.status(401).json({ message: 'Email missing' });

}

if (!password) {

return res.status(401).json({ message: 'Password missing' });

}

if (!ValidateEmail(email)) {

return res.status(401).json({ message: 'Invalid Email Address' });

}

const existingUser = await User.findOne({ email }).exec();

if (!existingUser) {

return res.status(401).json({ message: 'User does not exist' });

}

const isValid = await compareHashPassword(password, existingUser.password);

if (!isValid) {

return res.status(401).json({ message: 'Wrong Password' });

}

const user = {

email,

password,

};

jwt.sign({ user }, process.env.JWT\_SECRET, { expiresIn: '1h' }, (err, token) => {

if (err) {

console.error(err.message);

res.status(401).json({ message: 'Unable to Login' });

} else {

res.status(200).json({ token });

}

});

} catch (error) {

console.error(error.message);

res.status(401).json({ message: 'Unable to Login' });

}

});

module.exports = router;

**server.js**

const express = require('express');

const mongoose = require('mongoose');

const userRoute = require('./routes/user');

const blogRoute = require('./routes/blog');

const jwt = require('jsonwebtoken');

const cors = require('cors');

const User = require('./model/userSchema');

const Blog = require('./model/blogSchema');

require('dotenv').config();

const app = express();

app.use(cors()); // Enable CORS for all routes

app.use(express.json());

const connect = async () => {

try {

mongoose.connect(process.env.MONGO\_URL, {

useUnifiedTopology: true,

useNewUrlParser: true

});

console.log("Connected to mongodb");

} catch (error) {

console.log(`error while connecting mongodb : ${error}`);

}

}

connect();

// Routes

const verifyToken = (req, res, next) => {

const bearerHeader = req.headers['authorization'];

if (typeof bearerHeader !== 'undefined') {

const bearer = bearerHeader.includes(' ') ? bearerHeader.split(' ') : [null, bearerHeader];

const bearerToken = bearer[1];

jwt.verify(bearerToken, process.env.JWT\_SECRET, (err, data) => {

if (err) {

res.sendStatus(403).json({ message: "user not logged in" });

} else {

req.userData = data;

next();

}

});

} else {

res.sendStatus(403).json({ message: "user not logged in" });

}

};

app.use('/', userRoute);

app.use('/blog', verifyToken, blogRoute);

app.post('/getuser', verifyToken, async (req, res) => {

try {

const { userId } = req.body;

const userDetails = await User.findOne({ \_id: userId }).exec();

res.status(200).json({

fullName: `${userDetails?.firstName} ${userDetails?.lastName}`

});

} catch (error) {

console.log(error);

res.status(500).json({ message: 'Unable to find the user' });

}

})

app.get('/checkuser', verifyToken, async (req, res) => {

try {

const id = req.query.blogId;

const blogDetails = await Blog.findOne({ \_id: id }).exec();

const userId = blogDetails.author;

const userDetails = await User.findOne({ \_id: userId }).exec();

res.status(200).json({ email: userDetails.email });

} catch (error) {

console.log(error);

res.status(500).json({ message: 'Unable to find the user details' });

}

})

app.get('/', (req, res) => {

res.send("Welcome to ADT Assignment");

})

mongoose.connection.once('open', () => {

app.listen(8000, () => {

console.log("Server running on port 8000");

});

})

**Users collection**

**A screenshot of a computer

Description automatically generated**

**Blogs Collection**

**A screenshot of a computer

Description automatically generated**

**Comments Collection**

**A screenshot of a computer

Description automatically generated**

**Script to generate random users, blogs and comments**

const mongoose = require('mongoose');

const User = require('./model/userSchema');

const Blog = require('./model/blogSchema');

const Comment = require('./model/commentSchema');

const bcrypt = require("bcrypt");

const saltRounds = 10

const hashPassword = async (password) => {

try {

const hash = await bcrypt.hash(password, saltRounds);

return hash;

} catch (error) {

console.error(err.message);

}

}

function getRandomStringFromArray(stringArray) {

// Generate a random index between 0 and the length of the array

const randomIndex = Math.floor(Math.random() \* stringArray.length);

// Return the random string from the array

return stringArray[randomIndex];

}

const myArray = ["apple", "banana", "cherry", "date"];

async function populateData() {

try {

await mongoose.connect('mongodb://localhost:27017/adt', {

useUnifiedTopology: true,

useNewUrlParser: true

});

console.log("Connected to mongodb");

console.log("Populating the Data");

const newPassword = await hashPassword('password');

for (let i = 1; i < +100; i++) {

// sample users

const user1 = await User.create({

firstName: `Adam-${i}`,

lastName: `Lee-${i}`,

email: `john${i}@example.com`,

password: newPassword,

mobile: 8375022778,

dob: '1990-05-24'

});

getRandomStringFromArray(myArray);

// sample blog posts

const blog = await Blog.create({

title: `Sample Blog Title by Adam-${i} Lee-${i}`,

content: `Content of the blog given added by Adam-${i} Lee-${i}`,

author: user1.\_id,

authorName: `Adam-${i} Lee-${i}`,

tags: [getRandomStringFromArray(myArray), getRandomStringFromArray(myArray)],

categories: [getRandomStringFromArray(myArray), getRandomStringFromArray(myArray)]

});

// sample comments

const comment1 = await Comment.create({

text: `Great blog by Adam-${i} Lee-${i}`,

blogID: blog.\_id,

commenterName: `Adam-${i} Lee-${i}`

});

}

console.log("Date Populated");

mongoose.disconnect();

} catch (error) {

console.log(`error while connecting mongodb : ${error}`);

}

}

populateData();

**UI Screenshots**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a login form

Description automatically generated**