**Module – : Structuring REST API**

Q1. Create Rest API (get, post, delete, put, patch) for Task management.

===================================================================

1. Project Setup:

Create a new Node.js project directory.

Initialize a new Node.js project using npm init .

Install the required dependencies:

npm install express mongoose body-parser

=================================================================

2. MongoDB Connection:

Create a MongoDB connection file (e.g., db.js):

const mongoose = require('mongoose');

const dbURI = 'mongodb://localhost:27017/your\_database\_name'; // Replace with your MongoDB connection string

const connectDB = async () => {

try {

await mongoose.connect(dbURI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB connected');

} catch

(error) {

console.error('MongoDB connection error:', error);

process.exit(1);

}

};

module.exports = connectDB;

Import and call the connectDB function in your main server file.

==============================================================

3. Task Model:

Create a task.js file to define the Task schema:

const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({

title: {

type: String,

required: true,

},

description: {

type: String,

},

status: {

type: String,

enum: ['pending', 'in progress', 'completed'],

default: 'pending',

},

dueDate: {

type: Date,

},

});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

==============================================================

4. Server Setup:

Create a server file (e.g., index.js):

const express = require('express');

const bodyParser = require('body-parser');

const connectDB = require('./db');

const Task = require('./task');

const app = express();

const port = process.env.PORT || 3000;

app.use(bodyParser.json());

connectDB();

// GET all tasks

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

try {

const tasks = await Task.find();

res.json(tasks);

} catch (error) {

res.status(500).json({ error: 'Failed to fetch

tasks' });

}

});

// POST a new task

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

try {

const newTask = new Task(req.body);

await newTask.save();

res.status(201).json(newTask);

} catch (error)

{

res.status(400).json({ error: 'Invalid task data' });

}

});

// DELETE a task by ID

app.delete('/tasks/:id', async (req, res) => {

try {

const

deletedTask = await Task.findByIdAndDelete(req.params.id);

if (!deletedTask) {

return res.status(404).json({

error: 'Task not found' });

}

res.json({ message: 'Task deleted successfully' });

} catch (error) {

res.status(500).json({ error: 'Failed to delete task'

});

}

});

// PUT (update) a task by ID (replace entire object)

app.put('/tasks/:id', async (req, res) => {

try {

const updatedTask = await Task.findByIdAndUpdate(req.params.id, req.body, { new: true });

if (!updatedTask) {

return res.status(404).json({

error: 'Task not found' });

}

res.json(updatedTask);

} catch (error) {

res.status(400).json({ error:

'Invalid task data' });

}

});

// PATCH (update) a task by ID (partial update)

app.patch('/tasks/:id', async (req, res) => {

try {

const updatedTask = await Task.findByIdAndUpdate(req.params.id, req.body, { new: true });

if (!updatedTask) {

return res.status(404).json({

error: 'Task not found' });

}

res.json(updatedTask);

} catch (error) {

res.status(400).json({ error:

'Invalid task data' });

}

});

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

===============================================================

5. Start the Server:

Run node index.js in your terminal.

This code provides a solid foundation for task management REST APIs using Node.js and MongoDB. You can customize it further based on your specific requirements, such as adding more fields to the task model, implementing authentication and authorization, and enhancing error handling.

Q2. Create Rest API (get, post, delete, put, patch) for online shopping application.

1.Project Setup

Create a new Node.js project:

mkdir online-shopping

cd online-shopping

npm init -y

2.Install dependencies:

npm install express mongoose body-parser

3.MongoDB Connection

Create a MongoDB connection file (e.g., db.js):

const mongoose = require('mongoose');

const dbURI = 'mongodb://localhost:27017/online\_shopping'; // Replace with your MongoDB connection string

const connectDB = async () => {

try {

await mongoose.connect(dbURI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB connected');

} catch

(error) {

console.error('MongoDB connection error:', error);

process.exit(1);

}

};

module.exports = connectDB;

4.Models

Create models for products, categories, users, and orders:

// product.js

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

description: {

type: String,

},

price: {

type: Number,

required: true,

},

image: {

type: String,

},

category:

{

type: mongoose.Schema.Types.ObjectId,

ref: 'Category',

},

});

const Product = mongoose.model('Product', productSchema);

module.exports = Product;

// category.js

const mongoose = require('mongoose');

const categorySchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

});

const Category = mongoose.model('Category',

categorySchema);

module.exports = Category;

// user.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

email: {

type: String,

required:

true,

unique: true,

},

password: {

type: String,

required: true,

},

cart: {

type:

Array,

default: [],

},

});

const User = mongoose.model('User', userSchema);

module.exports = User;

// order.js

const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({

user: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

},

products: {

type: Array,

required: true,

},

total: {

type: Number,

required: true,

},

status: {

type: String,

enum: ['pending', 'processing', 'shipped', 'delivered'],

default: 'pending',

},

});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;

5.Server Setup

Create a server file (e.g., index.js):

const express = require('express');

const bodyParser = require('body-parser');

const connectDB = require('./db');

const Product = require('./product');

const Category = require('./category');

const User = require('./user');

const Order = require('./order');

const app = express();

const port = process.env.PORT || 3000;

app.use(bodyParser.json());

connectDB();

// Product routes

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

try {

const products = await Product.find();

res.json(products);

} catch (error) {

res.status(500).json({ error: 'Failed to fetch products'

});

}

});

// ... other product routes (post, get by id, update, delete)

// Category routes

// ...

// User routes

// ...

// Order routes

// ...

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});