FULL STACK WITH MERN (JAVA) ASSIGNMENT-3

Sanagala Sai Sreeja
Vignan's Nirula Institute of
Technology and Science
For Women (VNITSW)
4th B-Tceh (CSE)
20NN1A05A2

ASSIGNMENT-3: Create a RESTful API using express.js and create a database and index in MongoDB

1. INDEX.JS:

```
// index.js

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

const app = express();
const PORT = process.env.PORT || 4000;

// Middleware
app.use(bodyParser.json());

// Routes
const todoRoutes = require('./routes/todoRoutes');
app.use('/api/todos', todoRoutes);

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/todoDB', { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => {
    console.log('Connected to MongoDB');
    app.listen(PORT, () => {
      console.log(`Server is running on port ${PORT}`);
    });
  })
  .catch(err => console.error('Error connecting to MongoDB:', err));
```

2. ROUTES (todoRoutes.js):

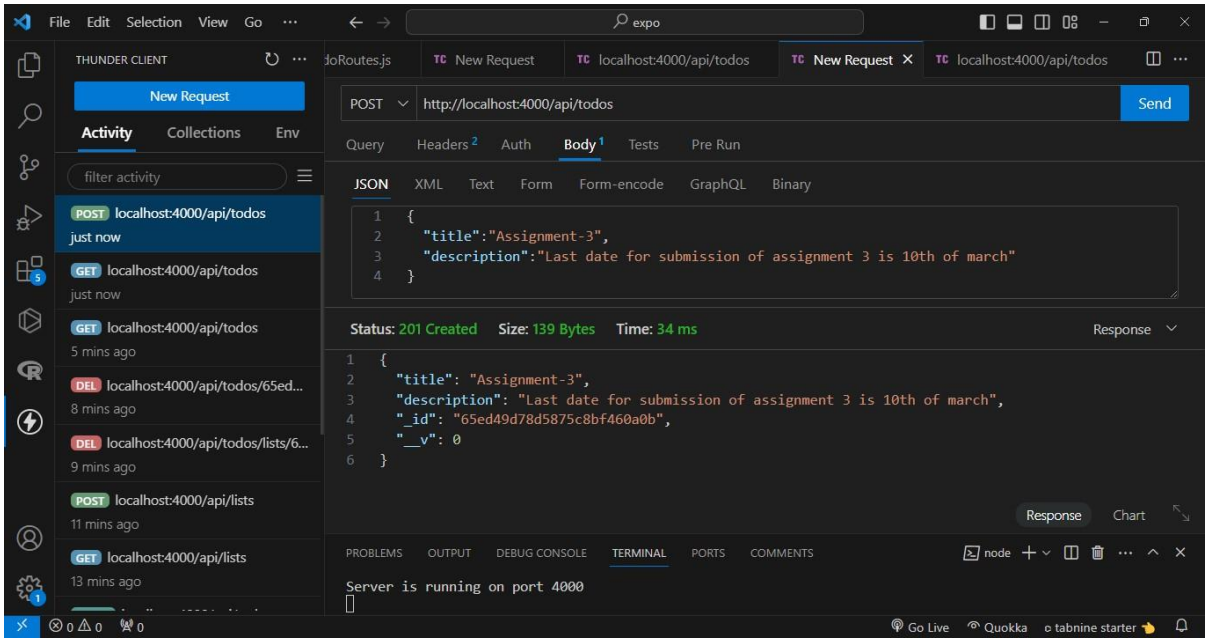
```
// routes/todoRoutes.js
const express = require('express');
const router = express.Router();
const Todo = require('../models/todoModel');

// Create a new todo
router.post('/', async (req, res) => {
  try {
    const todo = await Todo.create(req.body);
    res.status(201).json(todo);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

// Read all todos
router.get('/', async (req, res) => {
  try {
    const todos = await Todo.find();
    res.json(todos);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Update a todo
router.patch('/:id', async (req, res) => {
  try {
    const todo = await Todo.findByIdAndUpdate(req.params.id, req.body, { new:
true });
    res.json(todo);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
// Delete a todo
router.delete('/:id', async (req, res) => {
  try {
    await Todo.findByIdAndDelete(req.params.id);
    res.json({ message: 'Todo deleted' });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});


module.exports = router;
```

3. MODELS (todoModel.js):

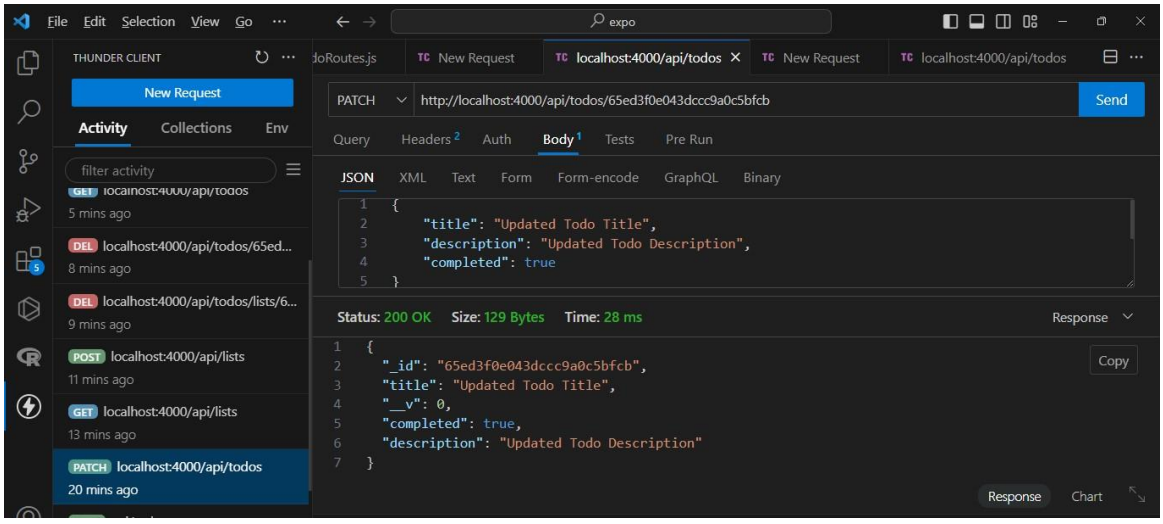
```
// models/todoModel.js

const mongoose = require('mongoose');

const todoSchema = new mongoose.Schema({
  title: String,
  description: String,
  completed: Boolean
});

const Todo = mongoose.model('Todo', todoSchema);

module.exports = Todo;
```

4. OUTPUT:

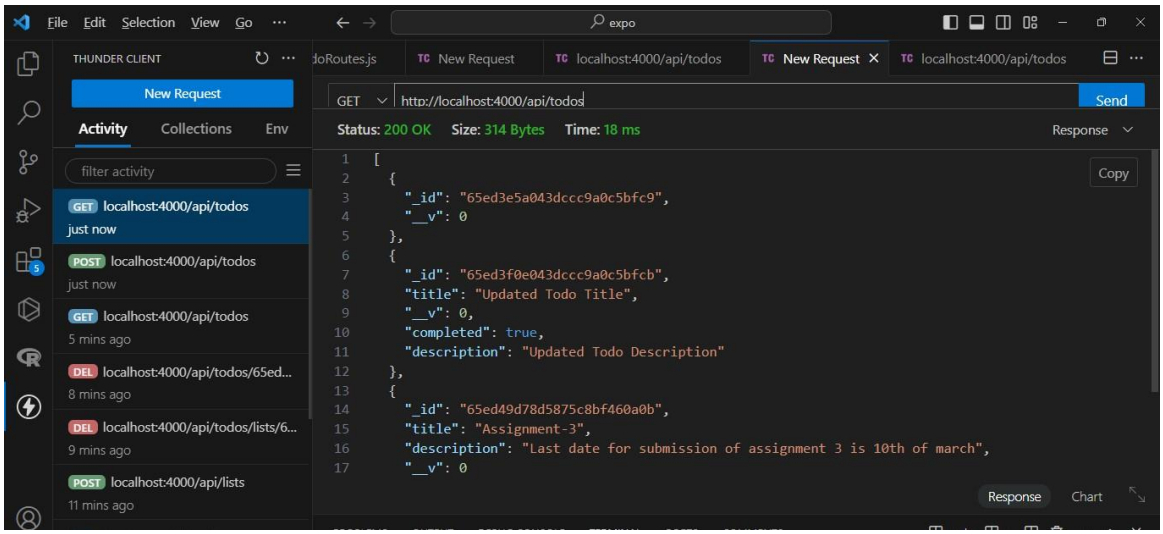
4.1 POST:



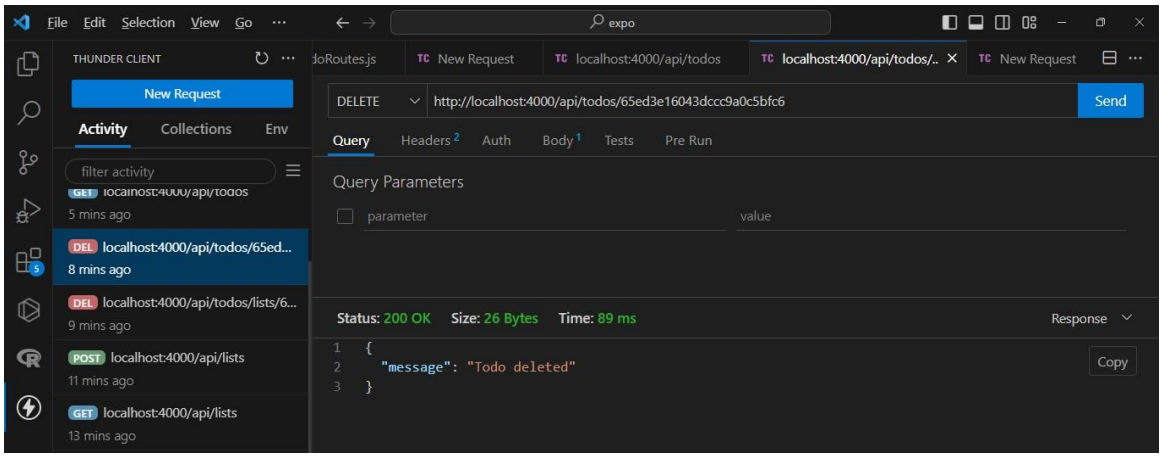
4.2 UPDATE:



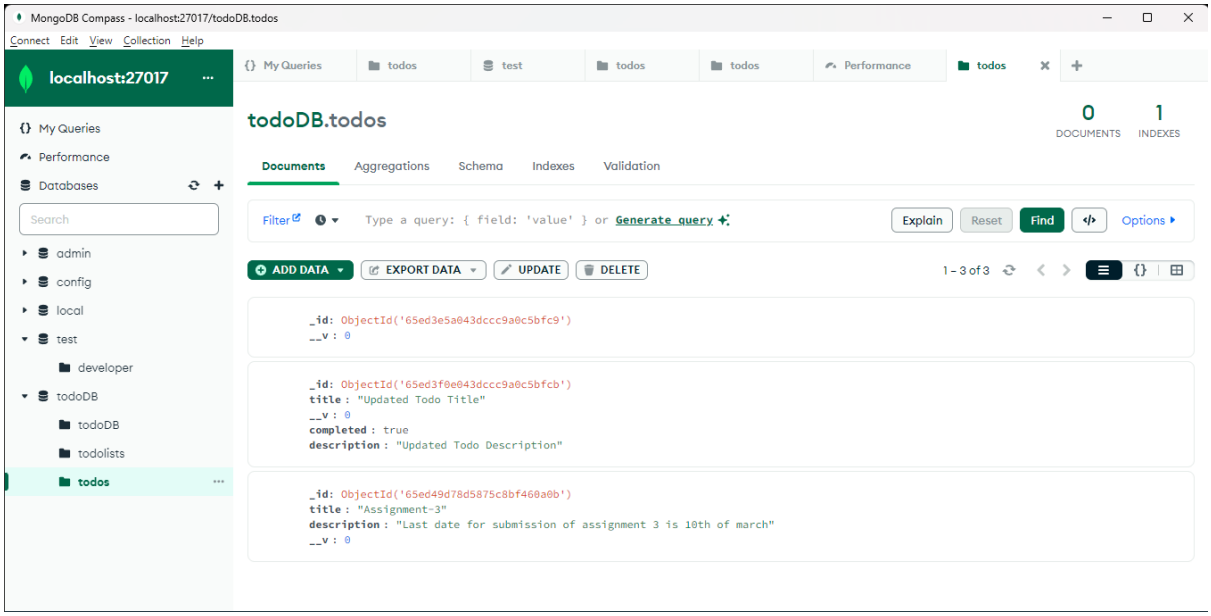
4.3 GET:



4.4 DELETE:



4.5 MONGODB:



5.EXPLANATION:

Initialize your project: Create a new directory for your project and initialize it with npm.

mkdir express-mongodb-api

cd express-mongodb-api

npm init -y

Install dependencies: Install necessary dependencies including Express.js, MongoDB driver, and any other required middleware.

npm install express mongoose body-parser

Set up your project structure: Create directories for your routes, models, and any other components you'll need.

Like- mkdir routes and models.

Objective:

Create a RESTful API for managing todo items.

Technologies Used:

MongoDB: A NoSQL database used to store todo items. Express.js: A Node.js framework used to create the API endpoints and handle HTTP requests. Node.js: A JavaScript runtime used for server-side development.

Assignment Components:

index.js: This file serves as the entry point of the application. It sets up the Express.js server, defines middleware (like bodyParser to parse JSON data), establishes routes, and connects to the MongoDB database.

todoRoutes.js: This file contains the route definitions for handling CRUD operations on todo items. It defines routes like POST (create), GET (read), PATCH (update), and DELETE (delete) for interacting with todo items.

todoModel.js: This file defines the schema for the todo items using Mongoose, a MongoDB object modeling tool for Node.js. It specifies the structure of todo items with fields like title, description, and completed.

Output:

The output section outlines the expected behavior of the API endpoints. It describes the actions to perform for each HTTP method (POST, GET, PATCH, DELETE) and provides example requests and responses.

POST /api/todos

Description: Creates a new todo item. Request Body: JSON object containing title, description, and completed fields. Response: Returns the created todo item with status code 201 (Created).

GET /api/todos

Description: Retrieves all todo items. Response: Returns an array of all todo items stored in the database.

PATCH /api/todos/:id

Description: Updates a todo item with the specified ID. Request Parameters: id (Todo item ID). Request Body: JSON object containing the fields to be updated (e.g., completed). Response: Returns the updated todo item.

DELETE /api/todos/:id

Description: Deletes a todo item with the specified ID. Request Parameters: id (Todo item ID). Response: Returns a message indicating the deletion of the todo item.