

# FULL STACK WITH MERN (JAVA) ASSIGNMENT-3

Sanagala Sai Sreeja  
Vignan's Nirula Institute of  
Technology and Science  
For Women (VNITSW)  
4<sup>th</sup> B-Tech (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');

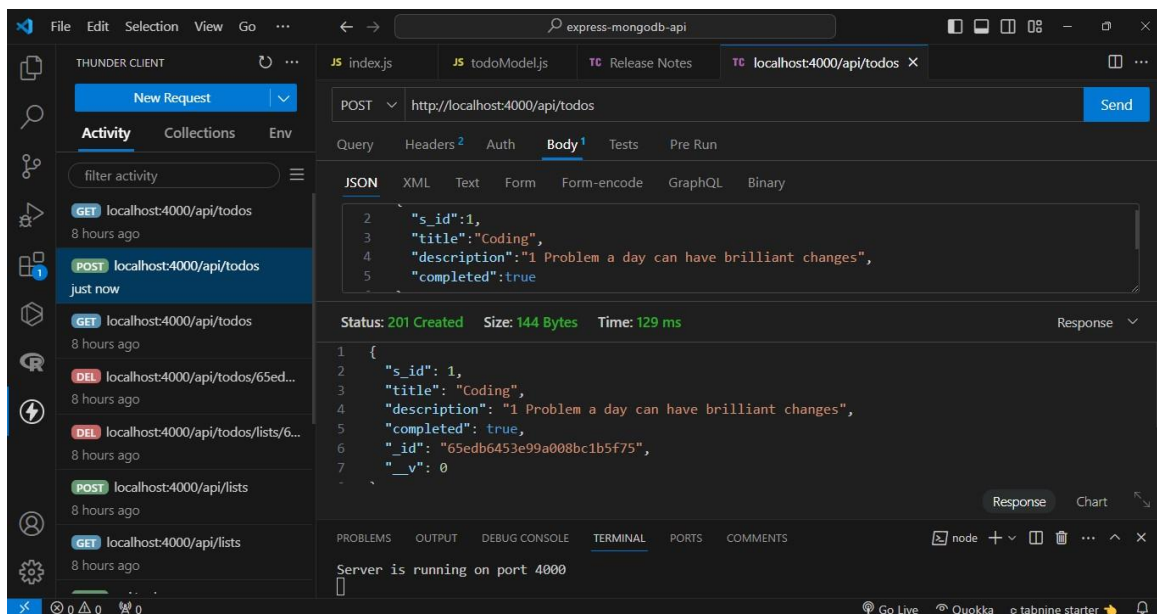
const todoSchema = new mongoose.Schema({
  s_id: Number,
  title: String,
  description: String,
  completed: Boolean
});

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

module.exports = Todo;
```

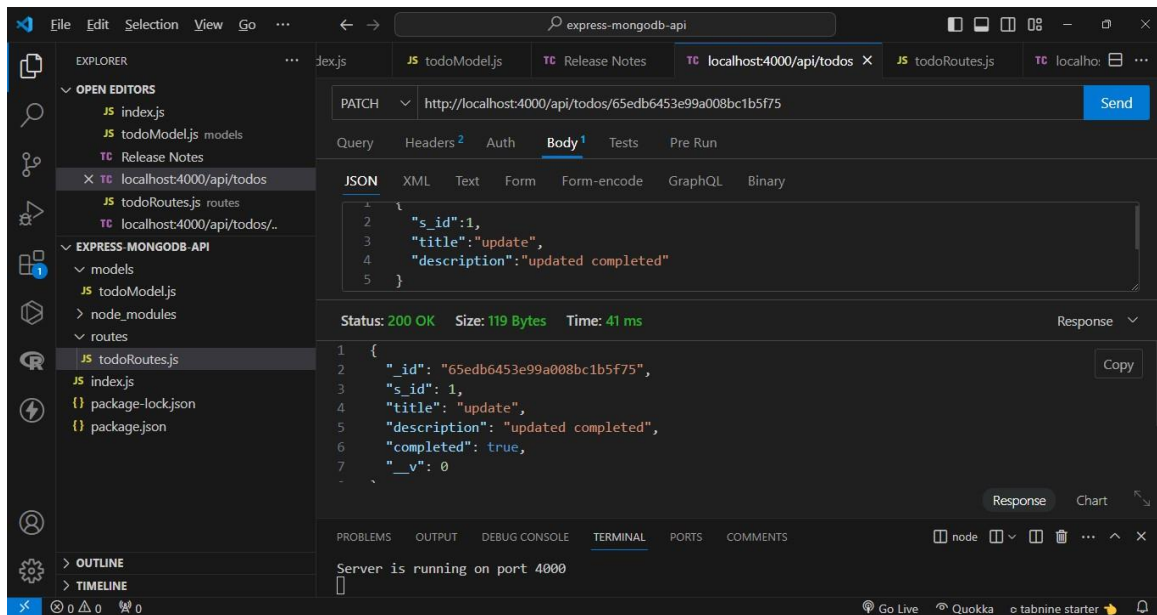
### 4. OUTPUT:

#### 4.1 POST:



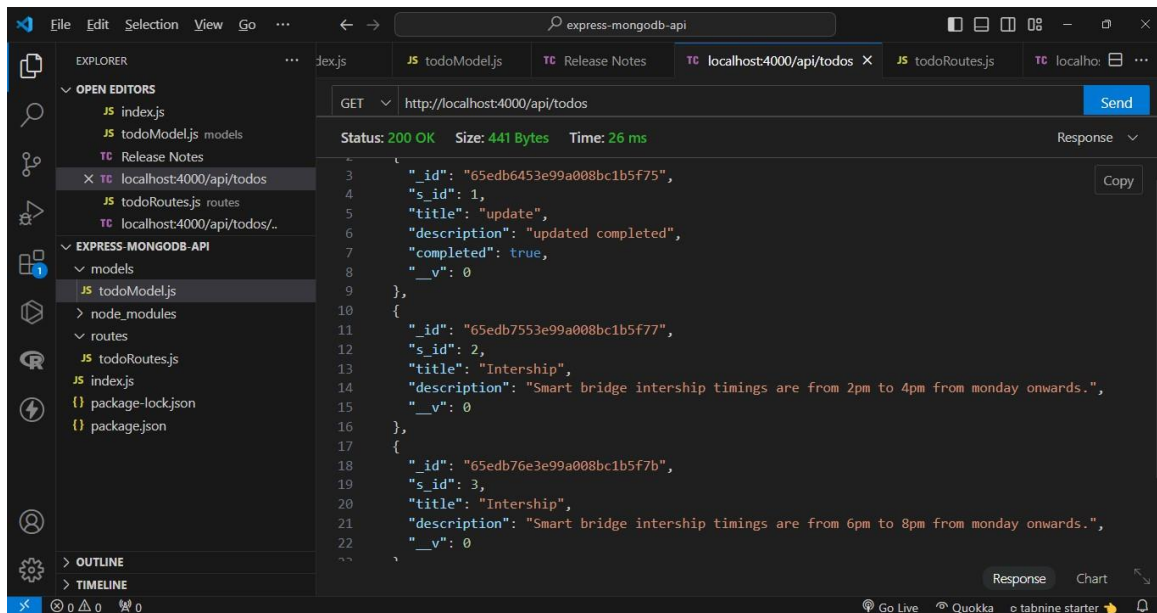
The screenshot displays the Thunder Client interface with a POST request to `http://localhost:4000/api/todos` successfully executed. The request body is a JSON object: `{ "s_id": 1, "title": "Coding", "description": "1 Problem a day can have brilliant changes", "completed": true }`. The response status is `201 Created` with a size of `144 Bytes` and a time of `129 ms`. The response body is a JSON object: `{ "s_id": 1, "title": "Coding", "description": "1 Problem a day can have brilliant changes", "completed": true, "_id": "65edb6453e99a008bc1b5f75", "__v": 0 }`. The terminal at the bottom shows the message `Server is running on port 4000`.

## 4.2 UPDATE:



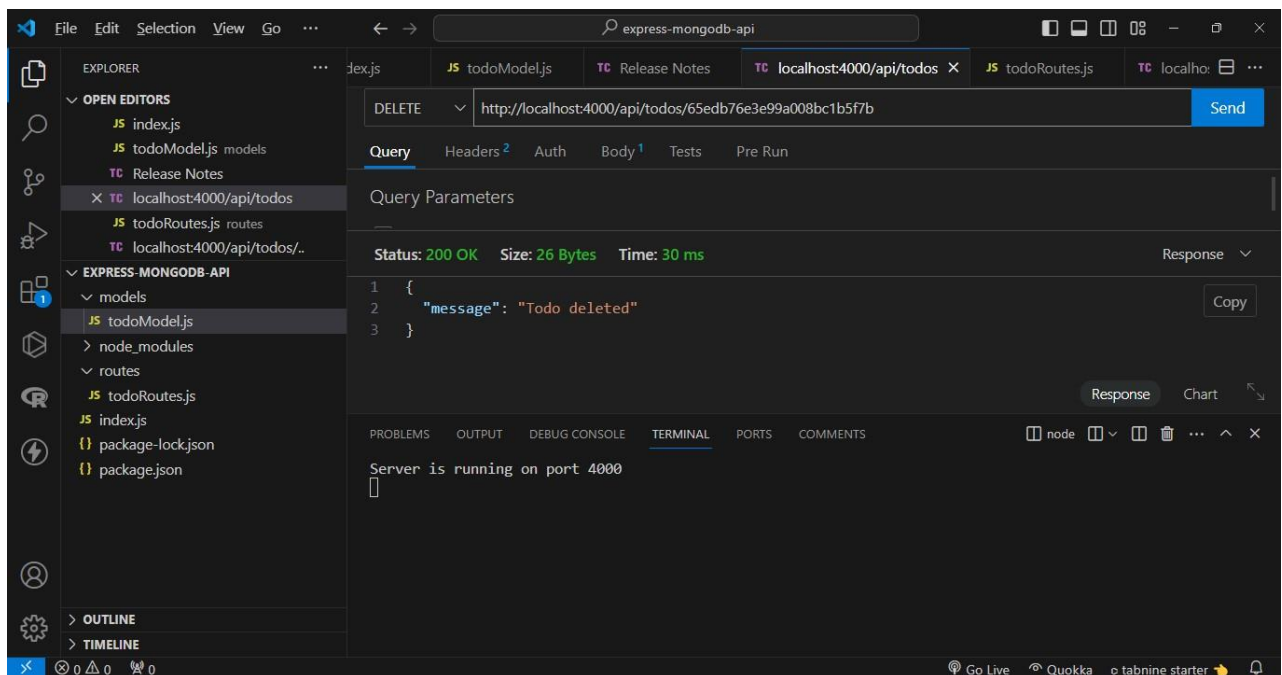
The screenshot shows the VS Code interface with the REST Client extension. A PATCH request is configured to `http://localhost:4000/api/todos/65edb6453e99a008bc1b5f75`. The request body is a JSON object: `{ "s_id": 1, "title": "update", "description": "updated completed" }`. The response is a 200 OK status with a size of 119 Bytes and a time of 41 ms. The response body is a JSON object: `{ "_id": "65edb6453e99a008bc1b5f75", "s_id": 1, "title": "update", "description": "updated completed", "completed": true, "__v": 0 }`. The Explorer sidebar shows the project structure, including `models`, `node_modules`, `routes`, `todoRoutes.js`, `index.js`, `package-lock.json`, and `package.json`.

## 4.3 GET:

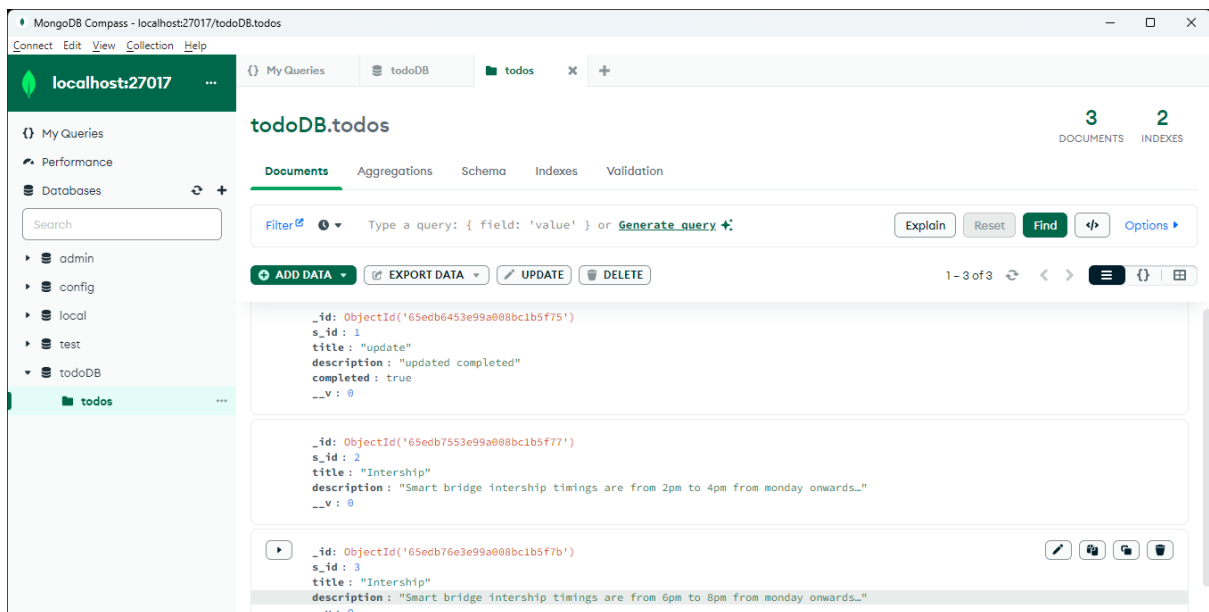


The screenshot shows the VS Code interface with the REST Client extension. A GET request is configured to `http://localhost:4000/api/todos`. The response is a 200 OK status with a size of 441 Bytes and a time of 26 ms. The response body is a JSON array of three todo items: `[ { "_id": "65edb6453e99a008bc1b5f75", "s_id": 1, "title": "update", "description": "updated completed", "completed": true, "__v": 0 }, { "_id": "65edb753e99a008bc1b5f77", "s_id": 2, "title": "Internship", "description": "Smart bridge internship timings are from 2pm to 4pm from monday onwards.", "__v": 0 }, { "_id": "65edb76e3e99a008bc1b5f7b", "s_id": 3, "title": "Internship", "description": "Smart bridge internship timings are from 6pm to 8pm from monday onwards.", "__v": 0 } ]`. The Explorer sidebar shows the project structure, including `models`, `node_modules`, `routes`, `todoRoutes.js`, `index.js`, `package-lock.json`, and `package.json`.

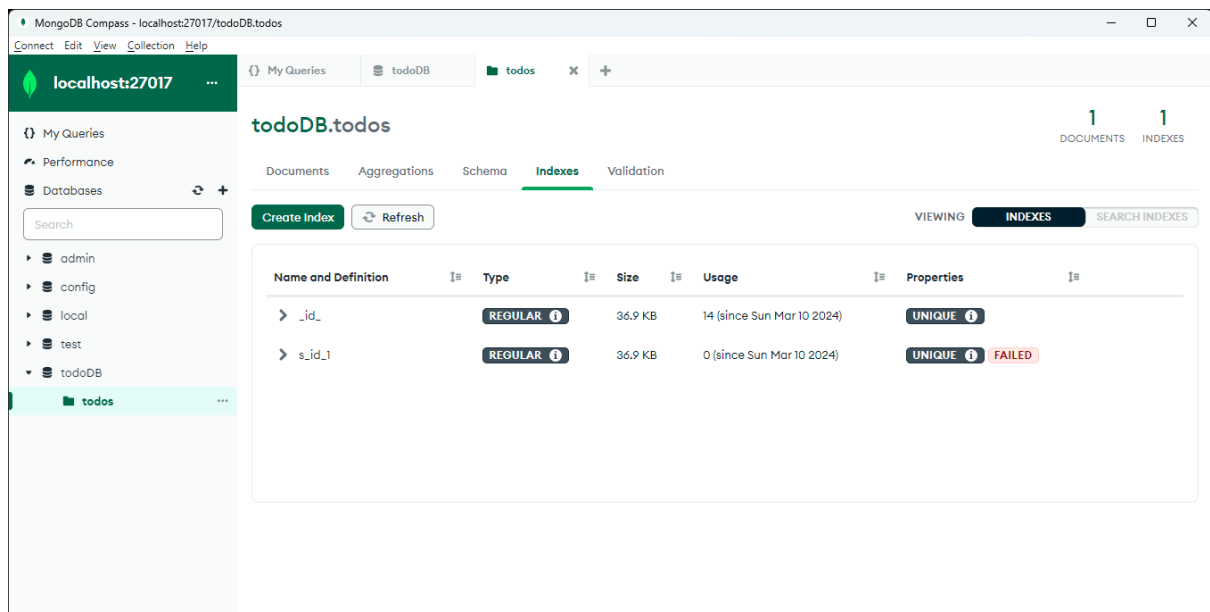
## 4.4 DELETE:



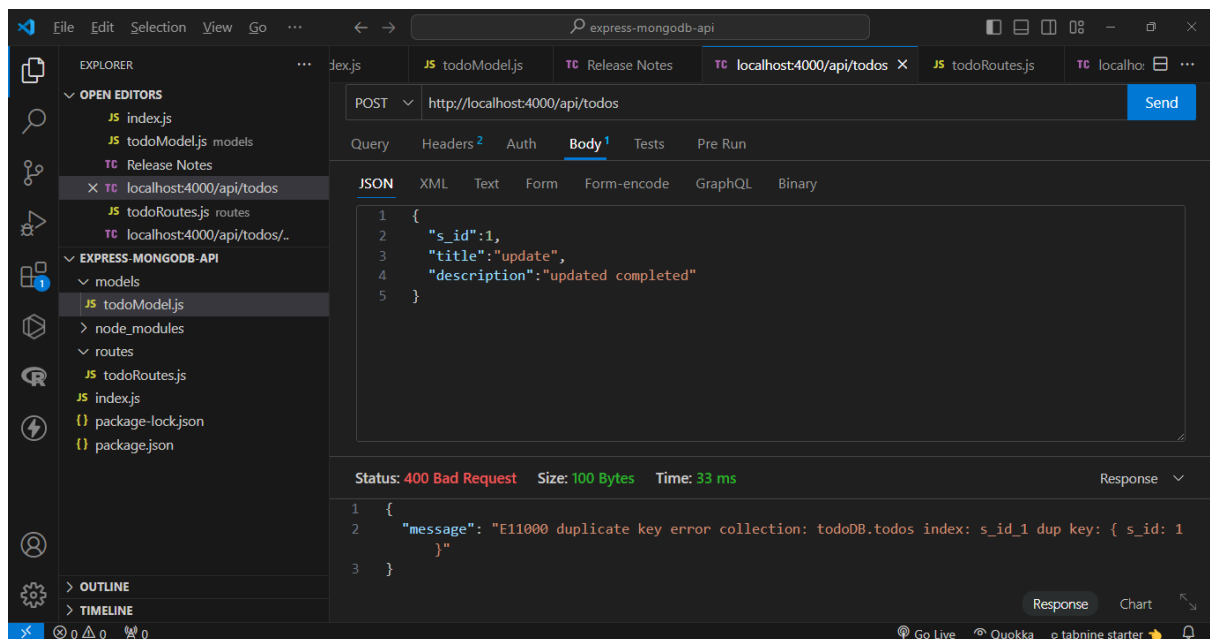
## 4.5 MONGODB:



## 4.6 INDEXES:



- As we created s\_id index as unique, so if we create a new document with same s\_id we will be getting the error message saying s\_id\_1 dup key: { s\_id: 1 }



## 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.

## **Indexes:**

Created a unique index named for s\_id so that only the unique s\_id todo list can be added. If we add any document of having same s\_id then it will show error message saying we can't add duplicate values to the s\_id index.

### **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.