

**6. Develop a CRUD-based dashboard for project management. Users can Create, Read, Update, and Delete tasks, and update their status (Pending, In Progress, Completed). Use MongoDB to store tasks and Node.js with Express to build the API.**

### app.js

```
const express = require('express');
const dotenv = require('dotenv').config();
const connectDB = require('./config/dbConnect');
const tasksRoutes = require('./routes/taskRoutes');
connectDB();
const app = express();
app.use(express.json());
// main routes
app.use("/api/tasks",tasksRoutes);
// catch-all route (for wrong URLs)
app.use((req, res) => {
res.status(404).json({ message: 'Route not found' });
});
app.listen(3000, ()=>{
console.log("server is running at port 3000");
})
```

### routes/taskRoutes.js

```
const express = require("express");
const router = express.Router();
const { getAllTasks,createTask,editTask,deleteTask} =
require('../controllers/taskController');

//router.route('/').get(getContact)
router.get('/', getAllTasks);
router.post('/',createTask);
router.put('/:id',editTask);
router.delete('/:id',deleteTask);

module.exports = router;
```

### config/dbConnect.js

```
const mongoose = require("mongoose");
const dotenv = require("dotenv").config()

const connectDB = async()=>{
    try{
        const connect = await mongoose.connect(process.env.CONNECTION_STRING);
        console.log("Database connected");
    }
}
```

```

        catch(err){

            console.log(err);
        }
    }
module.exports = connectDB;

```

### **models/taskModel.js**

```

const mongoose = require("mongoose");

const tasksSchema = mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: false
    },
    status: {
        type: String,
        enum: ['Pending', 'In Progress', 'Completed'],
        default: 'Pending'
    },
})
module.exports = mongoose.model("Task",tasksSchema);

```

### **controllers/taskController.js**

```

const Task = require("../models/taskModel");

const getAllTasks = async (req, res) => {
    try {
        // Find all tasks
        const tasks = await Task.find()
        res.status(200).json(tasks);
    }
    catch (error) {
        res.status(500).json({ message: 'Error retrieving tasks', error: error.message });
    }
}
const createTask = async (req, res) => {
    try {
        const {title,description,status} = req.body;
        if(!title || !description){

```

```

        res.status(400).json({message:"Title and description is mandatory"});
    }
    const task = await Task.create({
        title,
        description,
        status
    });
    res.status(201).json(task);
}
catch (error) {
    res.status(500).json({ message: 'Error retrieving tasks', error:
error.message });
}
}

const editTask = async (req, res) => {
    try {
        /* * Find the task by ID and update it.
         * { new: true } returns the modified document rather than the original.
         * { runValidators: true } ensures status enum rules are applied to the
update.
        */
        const task = await Task.findByIdAndUpdate(
            req.params.id,
            req.body,
            { new: true, runValidators: true }
        );

        if (!task) {
            return res.status(404).json({ message: 'Task not found for update' });
        }
        res.status(200).json(task);
    } catch (error) {
        res.status(400).json({ message: 'Error updating task', error:
error.message });
    }
}

const deleteTask = async (req, res) => {
    try {
        const task = await Task.findByIdAndDelete(req.params.id);

        if (!task) {
            return res.status(404).json({ message: 'Task not found for deletion' });
        }
        // Respond with status 204 (No Content) for successful deletion
        res.status(204).send();
    } catch (error) {

```

```

        res.status(500).json({ message: 'Error deleting task', error: error.message });
    }
}

module.exports = {
    getAllTasks,
    createTask,
    editTask,
    deleteTask
}

```

### .env

CONNECTION\_STRING="ATTACH YOUR MONGODB CONNECTION STRING HERE"

### OUTPUT:

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:3000/api/tasks`. Below the URL, the method is selected as `POST`. In the `Body` tab, the `JSON` tab is active, containing the following JSON payload:

```

1  {
2      "title": "Meeting",
3      "description": "Meeting at 4pm at COE Department"
4  }

```

At the bottom of the interface, the response is displayed under the `Body` tab. The status is `201 Created`, and the response body is:

```

1  {
2      "title": "Meeting",
3      "description": "Meeting at 4pm at COE Department",
4      "status": "Pending",
5      "_id": "68e3d9e4e6ffb39562cf0112",
6      "__v": 0
7  }

```

The screenshot shows the Postman application interface. At the top, there are tabs for 'GET Untitled' and 'POST http://'. The current tab is 'GET http://localhost:3000/api/tasks'. The status bar indicates 'No environment'. Below the tabs, the URL 'http://localhost:3000/api/tasks' is entered. There are 'Save' and 'Share' buttons. A 'Send' button is highlighted in blue. Below the URL, the method 'GET' is selected. The 'Cookies' tab is active. Under 'Query Params', there is a table with columns 'Key', 'Value', 'Descri...', and 'Bulk Edit'. The table is currently empty. The response status is '200 OK' with a duration of '53 ms' and a size of '654 B'. The response body is displayed as JSON:

```
1 [  
2 {  
3   "_id": "68e3d9e4e6ffb39562cf0112",  
4   "title": "Meeting",  
5   "description": "Meeting at 4pm at COE Departemnt",  
6   "status": "Pending",  
7   "__v": 0  
8 },  
9 {  
10   "_id": "68e3da68e6ffb39562cf0114",  
11   "title": "Notes",  
12   "description": "Notes of 5 modules to be prepared",  
13   "status": "Pending",  
14   "__v": 0  
15 },  
16 {  
17   "_id": "68e3da91e6ffb39562cf0116",  
18   "title": "Prepare Question Papers",
```

The screenshot shows two separate API requests in the Postman application.

**PUT Request:**

- URL: `http://localhost:3000/api/tasks/68e3d9e4e6ffb39562cf0112`
- Method: `PUT`
- Body (JSON):

```
1 {  
2   "status": "Completed"  
3 }
```
- Response: `200 OK`, 68 ms, 366 B. Body content:

```
1 {  
2   "_id": "68e3d9e4e6ffb39562cf0112",  
3   "title": "Meeting",  
4   "description": "Meeting at 4pm at COE Deparment",  
5   "status": "Completed",  
6   "__v": 0  
7 }
```

**DELETE Request:**

- URL: `http://localhost:3000/api/tasks/68e3da68e6ffb39562cf0114`
- Method: `DELETE`
- Body (Raw):

```
1
```
- Response: `204 No Content`, 107 ms, 134 B. Body content:

```
1
```