**Batch: A2 Roll No.: 16010122041**

**Experiment / assignment / tutorial No.\_\_9\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thurderclient |

**AIM:** Create a RESTful API server in Express and Node.js. Implementation + Testing application using postman/Thurderclient

**Problem Definition:**

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 3:** **Test the concepts and components of various front-end, back-end web app development technologies & frameworks using web development tools.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

**Testing in POSTMAN**

Testing with Postman is an essential part of the API development process, enabling developers to ensure that their endpoints function as intended. To begin, you set up Postman and create a new request by selecting the appropriate HTTP method, such as GET for retrieving data or POST for sending data to the server. You then input the API endpoint URL, which specifies the resource you want to interact with. For requests that involve data submission, like POST or PUT, you can include a JSON body in the request and set necessary headers, such as Content-Type: application/json, to inform the server about the format of the data being sent.

Once the request is configured, clicking the "Send" button dispatches it to the server, and Postman displays the response, including the status code, response time, and body content. This allows you to quickly identify if the API behaves as expected—checking for successful status codes (like 200 or 201) or error codes (like 400 or 500). Furthermore, Postman offers the ability to automate testing by writing JavaScript code within the "Tests" tab to validate responses programmatically, ensuring that your API returns the expected results for different scenarios. For example, you can assert that a successful response contains specific data or matches a certain structure. Additionally, organizing requests into collections allows for efficient management and collaboration, making it easy to share and document APIs for team use. Overall, Postman is a powerful tool that streamlines API testing and enhances the development workflow, making it easier to build reliable web services.

**Methodology:**

* **Setup Postman:** Install Postman on your machine, creating an account if needed.
* **Create a New Request:** Open Postman and create a new request. Choose the appropriate HTTP method (GET, POST, PUT, DELETE) depending on the action you want to perform on your API.
* **Set the URL:** Input the URL of your API endpoint. For example, if you're testing a user registration endpoint, it might be something like http://localhost:3000/api/users.
* **Add Headers:** Include necessary headers such as Content-Type: application/json for JSON data.
* **Input Request Body:** For POST or PUT requests, include the data you want to send to the server in the request body in JSON format.
* **Send the Request:** Click the "Send" button to dispatch the request to your server.
* **Analyze the Response:** Review the response received from the server, checking the status code and body for accuracy.
* **Automate Testing:** Utilize Postman's built-in testing tools to write scripts that can automate the testing of various endpoints and scenarios.

**Implementation Details:**

**Set Up the Project:**

* Initialize a new Node.js project using npm init -y.
* Install necessary dependencies

**Create Server File:**

* Create a file named server.js and set up a basic Express server

**Define API Routes:**

* Create routes for your API in a new file named routes.js

**Integrate Routes into Server:**

* Import and use the routes in your server.js file

**Database Connection:**

* If using MongoDB, connect to your database using Mongoose

**Testing Endpoints:**

* Use Postman to send requests to the defined endpoints. Test each method (GET, POST, etc.) and ensure the expected responses are received.

**Steps for execution:**

1. Ensure that MongoDB (if used) is running and your connection string is correctly set in the .env file.
2. Start the server by running:

node server.js

1. Open Postman and create requests to your API endpoints.
2. 8Verify the API's functionality by checking response codes and payloads.

**IMPLEMENTATION:**

# **REST API server and testing: App.js**

import React, { useState, useEffect } from 'react'; import './App.css';

function App() { const [newTask, setNewTask] = useState(''); const [updateTask, setUpdateTask] = useState(''); const [tasks, setTasks]

= useState([]);

const [editTask, setEditTask] = useState({ id: null, title: '' });

useEffect(()

=> {

fetchTasks()

;

}, []);

const fetchTasks = async () => { try {

const response = await fetch('http://localhost:5000/api/tasks'); if (response.*ok*) {

const data = await response.json(); setTasks(data);

} else {

console.error('Failed to fetch tasks. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

console.error('Error fetching tasks:', error);

}

};

const handleAddTask = async () => { try {

const response = await fetch('http://localhost:5000/api/tasks', { method: 'POST', headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ task: newTask })

});

if (response.*ok*) {

const data = await response.json(); setTasks([...tasks, data]);

setNewTask('');

} else {

console.error('Failed to add task. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

console.error('Error adding task:', error);

}

};

const handleUpdateTask = async () => { try {

const response = await fetch(`http://localhost:5000/api/tasks/${editTask.id}`, { method: 'PUT', headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ task: updateTask })

});

if (response.*ok*) {

const updatedTask = await response.json(); setUpdateTask(updatedTask.task); *// Update the updateTask state with the new value*

const updatedTasks = tasks.map((task) =>

task.\_id === updatedTask.\_id ? updatedTask : task

);

setTasks(updatedTasks); setEditTask({ id: null, title: '' });

} else {

console.error('Failed to update task. Server returned:', response.status, response.statusText);

}

} catch (error) {

console.error('Error updating task:', error);

}

};

const handleDeleteTask = async (id) => { try {

const response = await fetch(`http://localhost:5000/api/tasks/${id}`, { method: 'DELETE'

});

if (response.*ok*) {

*// If the deletion was successful on the server, update the state* const updatedTasks = tasks.filter(task

=> task.\_id !== id); setTasks(updatedTasks);

} else {

*// Handle server error or task not found*

console.error('Failed to delete task. Server returned:', response.*status*, response.*statusText*);

}

} catch (error) {

*// Handle network or other errors*

console.error('Error deleting task:', error);

}

};

return (

<div className="App">

<h1>Task List</h1>

<div>

<input type="tex t"

placeholder="Enter a task" value={newTask}

onChange={(e) => setNewTask(e.target.value)}

/>

<button onClick={handleAddTask}>Add Task</button>

</div>

<ul>

{tasks.map((task) => (

<li key={task.\_id}>

{task.task}{' '}

<button onClick={() => handleDeleteTask(task.\_id)}>Delete</button>

<button onClick={() => { setEditTask({ id: task.\_id, title:

task.task }); setUpdateTask(task.task);

}}>Edit</button>

{editTask.id === task.\_id && (

<div>

<input type="tex t"

placeholder="Enter updated task" value={updateTask}

onChange={(e) => setUpdateTask(e.target.value)}

/>

<button onClick={handleUpdateTask}>Update</button>

</div>

)}

</li>

))}

</ul>

</div>

);

}

export default App;

} else {

*// Handle server error or task not found*

console .error ('Failed to delete task. Server returned:' , response .*status* ,response .*statusText* );

}

} catch (error) {

*// Handle network or other errors*

console .error ('Error deleting task:' , error);

}

};

return (

<div className ="App" >

<h1>Task List</h1>

<div>

<input type="t ext"

placeholder ="Enter a task" value={newTask }

onChange ={(e) => setNewTask (e.target .value) }

/>

<button onClick ={handleAddTask }>Ad Task</button>

</div>

<ul>

{tasks .ma ((task) => (

<li key={task.\_id}>

{task.task}{' '}

<button onClick ={() => handleDeleteTask (task.\_id)}>Delete </button>

<button onClick ={() => { setEditTask ({ id: task.\_id, title:

task.task });setUpdateTask (task.task);

}}>Edit</button>

{editTask .id === task.\_id && (

<div>

<input type="t ext"

placeholder ="Enter updated task" value={updateTask }

onChange ={(e) => setUpdateTask (e.target .value) }

/>

# **Server.js**

const express = require('express');

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

const dotenv = require('dotenv') dotenv.config();

const app = express(); const PORT = 5000;

const uri = process.env.MONGODB\_URI;

app.use(cors()); app.use(bodyParser.json());

mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true

});

const taskSchema = new mongoose.Schema({ task: { type:

String, required: true

}

});

const Task = mongoose.model('Task', taskSchema); app.post('/api/tasks', async (req, res) => {

const { task } = req.body;

const newTask = new Task({ task

}); await newTask.save(); res.json(newTask);

});

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

const tasks = await Task.find(); res.json(tasks);

});

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

{ const { id } = req.params; const { task

} = req.body; try {

const updatedTask = await Task.findByIdAndUpdate(id, { task }, { new: true }); if (!updatedTask) {

return res.status(404).json({ message: 'Task not found' });

}

res.json(updatedTask);

} catch (error) {

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

'Internal server error' });

}

});

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

{ const { id } = req.params; try { const deletedTask = await

Task.findByIdAndDelete(id); if (!deletedTask)

{

console.log(`Task with ID ${id} not found.`);

return res.status(404).json({ message: 'Task not found' });

}

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

} catch (error) {

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

}

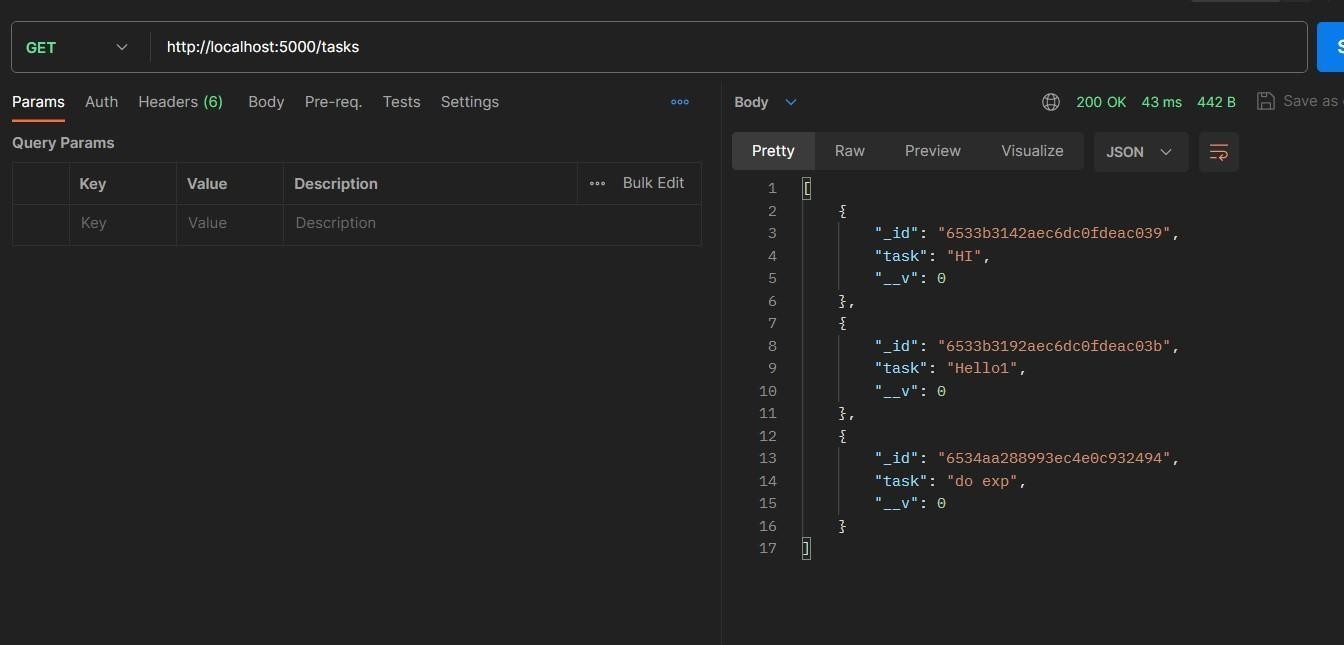
});

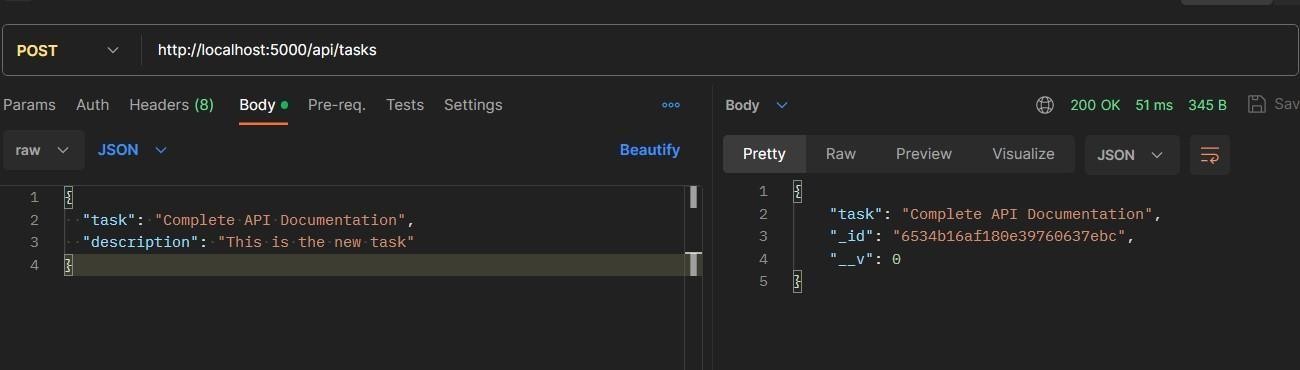
app.listen(PORT, () => {

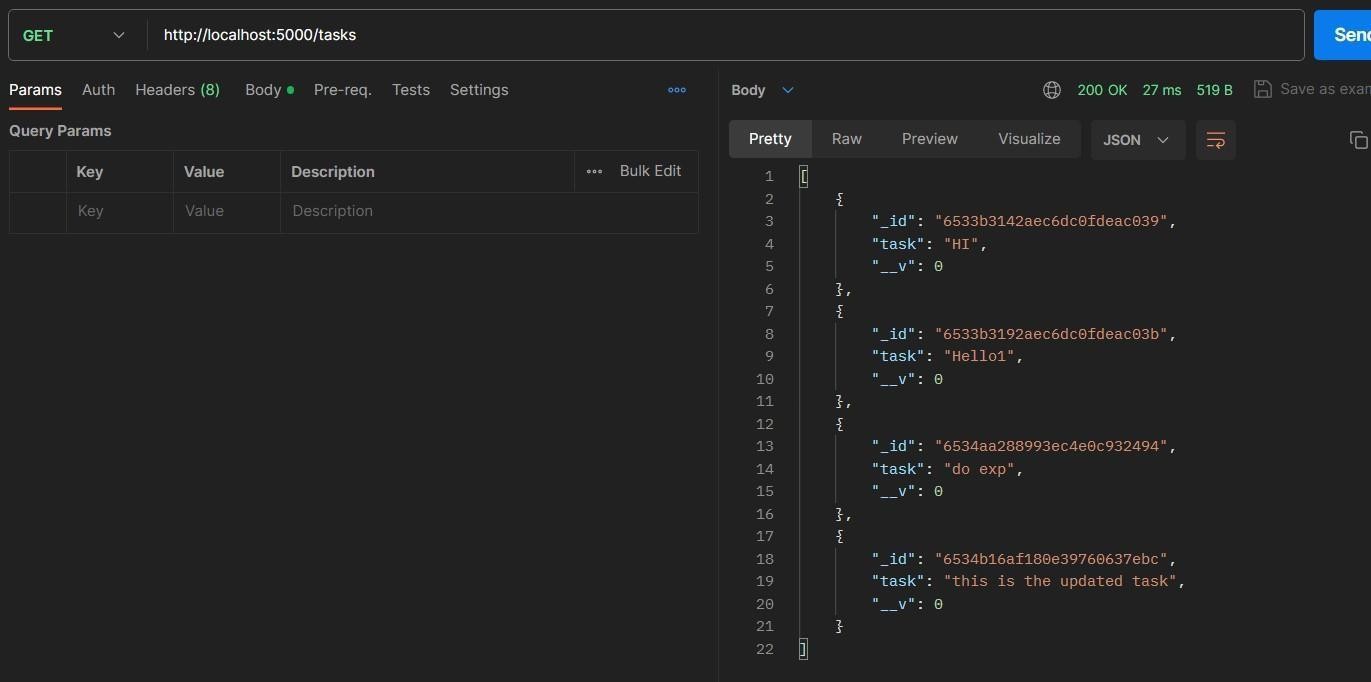
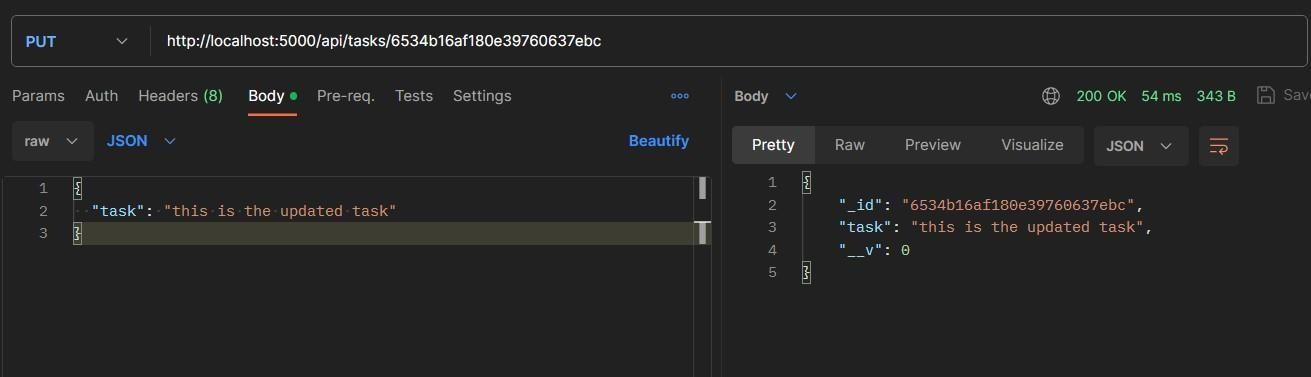
console.log(`Server is running on port ${PORT}`); });

**Output:**

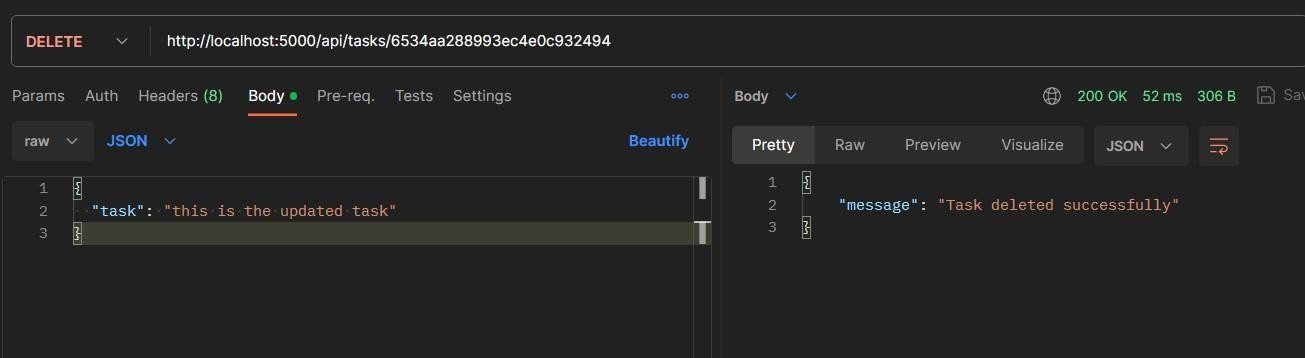
**Read/GET all:**

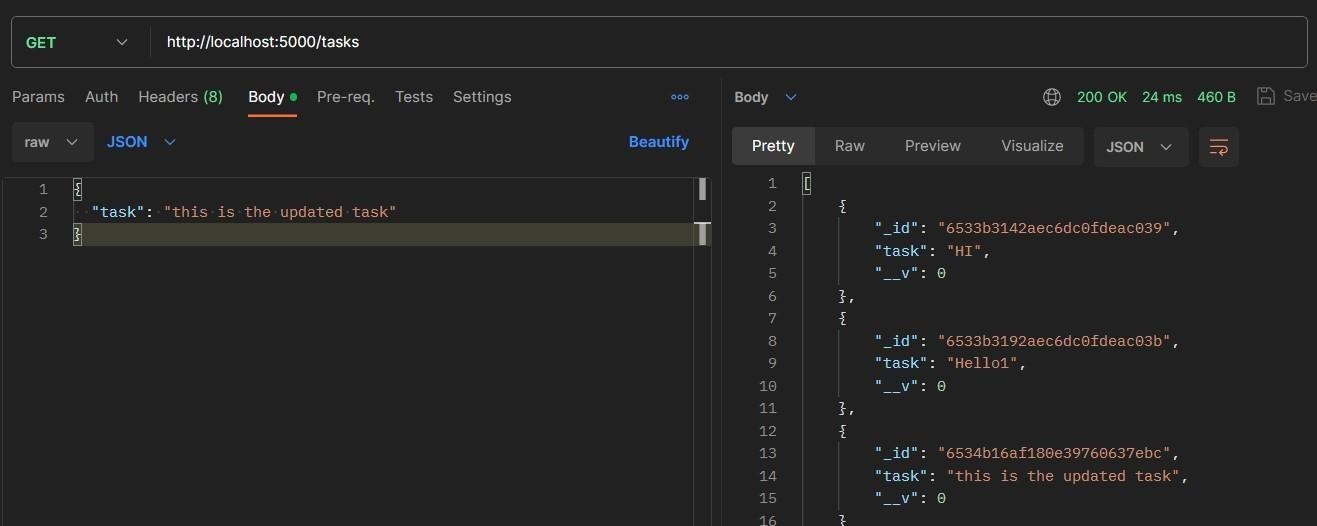


**Create: -**

**Update: -**

**Delete: -**





**Conclusion:** Learnt and implemented RestAPI using Postman.