**Backend**

**1. Setup Backend with Node.js and Express**

**a. Initialize Project**

bash

Copy code

mkdir tree-visualizer-backend

cd tree-visualizer-backend

npm init -y

npm install express mongoose body-parser cors

**b. Create server.js**

javascript

Copy code

const express = require('express');

const mongoose = require('mongoose');

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

const cors = require('cors');

const app = express();

app.use(bodyParser.json());

app.use(cors());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/treeDB', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.log(err));

// Routes

const treeRoutes = require('./routes/treeRoutes');

app.use('/api/tree', treeRoutes);

const PORT = 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**2. Create Models**

**a. TreeNode Schema (common for all tree types)**

javascript

Copy code

const mongoose = require('mongoose');

const treeNodeSchema = new mongoose.Schema({

value: Number,

left: { type: mongoose.Schema.Types.ObjectId, ref: 'TreeNode' },

right: { type: mongoose.Schema.Types.ObjectId, ref: 'TreeNode' }

});

module.exports = mongoose.model('TreeNode', treeNodeSchema);

**3. Create Routes**

**a. routes/treeRoutes.js**

javascript

Copy code

const express = require('express');

const router = express.Router();

const TreeNode = require('../models/treeNode');

// Get Tree

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

try {

const rootNode = await TreeNode.findOne(); // Example: fetch root node

res.json(rootNode);

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Add Node (BST logic example)

router.post('/add', async (req, res) => {

const { value } = req.body;

try {

let current = await TreeNode.findOne(); // assuming one root

if (!current) {

const newNode = new TreeNode({ value });

await newNode.save();

return res.json(newNode);

}

while (true) {

if (value < current.value) {

if (!current.left) {

const newNode = new TreeNode({ value });

current.left = newNode.\_id;

await newNode.save();

await current.save();

return res.json(newNode);

}

current = await TreeNode.findById(current.left);

} else {

if (!current.right) {

const newNode = new TreeNode({ value });

current.right = newNode.\_id;

await newNode.save();

await current.save();

return res.json(newNode);

}

current = await TreeNode.findById(current.right);

}

}

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Delete Tree Node

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

try {

await TreeNode.findByIdAndDelete(req.params.id);

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

} catch (err) {

res.status(500).json({ error: err.message });

}

});

module.exports = router;

**Frontend**

**1. Setup React App**

**a. Initialize Project**

bash

Copy code

npx create-react-app tree-visualizer-frontend

cd tree-visualizer-frontend

npm install axios react-d3-tree

**2. Create API Service**

Create an api.js file for managing API requests.

**src/api.js**

javascript

Copy code

import axios from 'axios';

const API = axios.create({ baseURL: 'http://localhost:5000/api/tree' });

export const getTree = () => API.get('/');

export const addNode = (value) => API.post('/add', { value });

export const deleteNode = (id) => API.delete(`/${id}`);

**3. Create TreeVisualizer Component**

**src/components/TreeVisualizer.js**

javascript

Copy code

import React, { useEffect, useState } from 'react';

import Tree from 'react-d3-tree';

import { getTree, addNode, deleteNode } from '../api';

const TreeVisualizer = () => {

const [treeData, setTreeData] = useState(null);

useEffect(() => {

fetchTreeData();

}, []);

const fetchTreeData = async () => {

try {

const { data } = await getTree();

setTreeData(formatTree(data));

} catch (err) {

console.error(err);

}

};

const formatTree = (node) => {

if (!node) return null;

return {

name: node.value.toString(),

children: [formatTree(node.left), formatTree(node.right)].filter(Boolean)

};

};

const handleAddNode = async () => {

const value = parseInt(prompt('Enter node value:'), 10);

if (!isNaN(value)) {

await addNode(value);

fetchTreeData();

}

};

const handleDeleteNode = async () => {

const id = prompt('Enter node ID to delete:');

if (id) {

await deleteNode(id);

fetchTreeData();

}

};

return (

<div>

<button onClick={handleAddNode}>Add Node</button>

<button onClick={handleDeleteNode}>Delete Node</button>

<div id="treeWrapper" style={{ width: '100%', height: '500px' }}>

{treeData && <Tree data={treeData} orientation="vertical" />}

</div>

</div>

);

};

export default TreeVisualizer;

**4. Integrate in App**

Update App.js:

javascript

Copy code

import React from 'react';

import TreeVisualizer from './components/TreeVisualizer';

function App() {

return (

<div className="App">

<h1>Tree Visualizer</h1>

<TreeVisualizer />

</div>

);

}

export default App;

**Full Flow Summary**

1. **Backend**:
   * MongoDB stores the tree nodes.
   * API routes handle tree operations (fetch, add, delete).
   * Tree logic (BST/AVL rules) applied in route handlers.
2. **Frontend**:
   * **TreeVisualizer** component fetches tree data.
   * Renders tree using react-d3-tree.
   * Add/Delete node buttons trigger backend updates via API.