**Q5: File System Management (Tree)**: Simulate a **file system** where directories and files are stored in a **tree structure**. Implement operations like creating new files, deleting files, and listing files in different traversal orders (pre-order, post-order, in-order).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>File System Management (Tree)</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

#tree-view {

margin: 20px 0;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

background-color: #f9f9f9;

}

.node {

margin-left: 20px;

}

button {

margin-top: 10px;

}

</style>

</head>

<body>

<h1>File System Management (Tree)</h1>

<div>

<label for="node-name">Node Name:</label>

<input type="text" id="node-name" placeholder="Enter file/folder name">

<button onclick="createNode()">Create</button>

<button onclick="deleteNode()">Delete</button>

</div>

<div>

<h3>Traversal Options:</h3>

<button onclick="traverseTree('preorder')">Pre-order</button>

<button onclick="traverseTree('postorder')">Post-order</button>

<button onclick="traverseTree('inorder')">In-order</button>

</div>

<div>

<h3>Traversal Output:</h3>

<div id="traversal-output"></div>

</div>

<div id="tree-view">

<strong>Tree Structure:</strong>

<div id="root" class="node">Root</div>

</div>

<script>

class TreeNode {

constructor(name) {

this.name = name;

this.children = [];

}

}

// Tree Structure

const root = new TreeNode('Root');

let currentNode = root; // Pointer to the current node

function createNode() {

const nodeName = document.getElementById('node-name').value.trim();

if (!nodeName) {

alert('Please enter a valid name.');

return;

}

const newNode = new TreeNode(nodeName);

currentNode.children.push(newNode);

updateTreeView();

}

function deleteNode() {

if (currentNode === root) {

alert('Cannot delete the root node.');

return;

}

const parent = findParent(root, currentNode);

if (parent) {

parent.children = parent.children.filter(child => child !== currentNode);

currentNode = root; // Reset current node to root after deletion

updateTreeView();

}

}

function findParent(node, target) {

for (let child of node.children) {

if (child === target) return node;

const result = findParent(child, target);

if (result) return result;

}

return null;

}

function traverseTree(order) {

let output = [];

if (order === 'preorder') preorder(root, output);

else if (order === 'postorder') postorder(root, output);

else if (order === 'inorder') inorder(root, output);

document.getElementById('traversal-output').innerText = output.join(' -> ');

}

function preorder(node, output) {

if (!node) return;

output.push(node.name);

for (let child of node.children) preorder(child, output);

}

function postorder(node, output) {

if (!node) return;

for (let child of node.children) postorder(child, output);

output.push(node.name);

}

function inorder(node, output) {

if (!node || node.children.length === 0) {

if (node) output.push(node.name);

return;

}

const mid = Math.floor(node.children.length / 2);

for (let i = 0; i < mid; i++) inorder(node.children[i], output);

output.push(node.name);

for (let i = mid; i < node.children.length; i++) inorder(node.children[i], output);

}

function updateTreeView() {

const treeView = document.getElementById('tree-view');

treeView.innerHTML = '<strong>Tree Structure:</strong>';

renderTree(root, treeView);

}

function renderTree(node, container) {

const div = document.createElement('div');

div.className = 'node';

div.textContent = node.name;

div.onclick = (e) => {

e.stopPropagation();

currentNode = node;

alert(`Current node: ${node.name}`);

};

container.appendChild(div);

for (let child of node.children) {

renderTree(child, div);

}

}

updateTreeView(); // Initialize tree view

</script>

</body>

</html>