**Q1: Organizational Hierarchy Management System**: Implement an **organization’s hierarchy** using a **tree structure** where each node represents an employee. Simulate promotions, new hires, and removals dynamically, ensuring the tree stays balanced.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Organizational Hierarchy</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

#tree-container {

margin-top: 20px;

}

ul {

list-style-type: none;

padding-left: 20px;

}

li {

margin: 10px 0;

cursor: pointer;

}

.node {

display: inline-block;

margin: 5px 0;

padding: 8px 12px;

border: 1px solid #ccc;

border-radius: 4px;

background-color: #f9f9f9;

}

.controls {

margin-top: 20px;

}

.controls input {

margin-right: 10px;

}

</style>

</head>

<body>

<h1>Organizational Hierarchy Management</h1>

<div class="controls">

<input type="text" id="employee-name" placeholder="Employee Name">

<button onclick="addEmployee()">Add Employee</button>

<button onclick="promoteEmployee()">Promote</button>

<button onclick="removeEmployee()">Remove</button>

</div>

<h3>Hierarchy:</h3>

<div id="tree-container">

<ul id="tree-root"></ul>

</div>

<script>

const root = { name: "CEO", children: [] }; // Initial hierarchy root

let selectedNode = root;

function renderTree(node, parentElement) {

parentElement.innerHTML = ""; // Clear existing tree

const listItem = document.createElement("li");

listItem.className = "node";

listItem.textContent = node.name;

// Mark the selected node

if (node === selectedNode) {

listItem.style.backgroundColor = "#cce5ff";

}

listItem.onclick = (e) => {

e.stopPropagation(); // Prevent propagation to parent nodes

selectedNode = node;

renderTree(root, document.getElementById("tree-root"));

};

const childrenContainer = document.createElement("ul");

node.children.forEach((child) => renderTree(child, childrenContainer));

listItem.appendChild(childrenContainer);

parentElement.appendChild(listItem);

}

function addEmployee() {

const name = document.getElementById("employee-name").value.trim();

if (name) {

selectedNode.children.push({ name, children: [] });

renderTree(root, document.getElementById("tree-root"));

document.getElementById("employee-name").value = ""; // Clear input

} else {

alert("Please enter a valid name!");

}

}

function promoteEmployee() {

if (selectedNode === root) {

alert("Cannot promote the root node!");

return;

}

const parent = findParent(root, selectedNode);

if (parent) {

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

selectedNode.children.push(...parent.children);

parent.children = [selectedNode];

renderTree(root, document.getElementById("tree-root"));

}

}

function removeEmployee() {

if (selectedNode === root) {

alert("Cannot remove the root node!");

return;

}

const parent = findParent(root, selectedNode);

if (parent) {

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

renderTree(root, document.getElementById("tree-root"));

}

}

function findParent(current, target) {

if (current.children.includes(target)) {

return current;

}

for (const child of current.children) {

const found = findParent(child, target);

if (found) return found;

}

return null;

}

// Initial render

renderTree(root, document.getElementById("tree-root"));

</script>

</body>

</html>