NAME: JANHVI SINGH

REG NO: 22BCE8544

MOD 8

HTML CODE:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>LaunchDarkly Employee Portal</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 20px;

            background-color: #f4f4f4;

            color: #333;

        }

        h1, h2 {

            color: #0056b3;

        }

        #controls {

            margin-bottom: 20px;

            padding: 15px;

            background-color: #e9ecef;

            border-radius: 8px;

            display: flex;

            gap: 10px;

        }

        .employee-list {

            display: grid;

            grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));

            gap: 20px;

            margin-top: 20px;

        }

        .employee-card {

            background-color: #fff;

            border: 1px solid #ddd;

            border-radius: 8px;

            padding: 20px;

            box-shadow: 0 2px 4px rgba(0,0,0,0.1);

        }

        .employee-card h3 {

            margin-top: 0;

            color: #333;

        }

        .employee-card p {

            margin: 5px 0;

            font-size: 0.9em;

        }

        .status {

            font-weight: bold;

            display: inline-block;

            padding: 3px 8px;

            border-radius: 4px;

            font-size: 0.8em;

        }

        .status.active {

            background-color: #d4edda;

            color: #155724;

        }

        .status.inactive {

            background-color: #f8d7da;

            color: #721c24;

        }

        .toggle-status-btn {

            background-color: #007bff;

            color: white;

            border: none;

            padding: 8px 12px;

            border-radius: 5px;

            cursor: pointer;

            margin-top: 10px;

            font-size: 0.85em;

        }

        .toggle-status-btn:hover {

            background-color: #0056b3;

        }

        .toggle-status-btn:active {

            background-color: #004085;

        }

    </style>

</head>

<body>

    <h1>LaunchDarkly Employee Portal</h1>

    <div id="controls">

        <button id="showAllBtn">Show All Employees</button>

        <button id="showActiveBtn">Show Only Active Employees</button>

        <button id="sortByNameBtn">Sort by Name</button>

    </div>

    <h2>Employee List</h2>

    <div id="employee-list" class="employee-list">

        <p>Loading employee data...</p>

    </div>

    <script src="employees.js"></script>

</body>

</html>

JS CODE:

// Define the array of employee objects

let employees = [

    { id: 1, name: 'Alice Johnson', position: 'Software Engineer', department: 'Engineering', isActive: true },

    { id: 2, name: 'Bob Williams', position: 'Product Manager', department: 'Product', isActive: false },

    { id: 3, name: 'Charlie Brown', position: 'UX Designer', department: 'Design', isActive: true },

    { id: 4, name: 'Diana Miller', position: 'QA Engineer', department: 'Engineering', isActive: true },

    { id: 5, name: 'Eve Davis', position: 'HR Specialist', department: 'Human Resources', isActive: false },

    { id: 6, name: 'Frank White', position: 'Sales Manager', department: 'Sales', isActive: true },

    { id: 7, name: 'Grace Taylor', position: 'Marketing Coordinator', department: 'Marketing', isActive: true }

];

const employeeListDiv = document.getElementById('employee-list');

const showAllBtn = document.getElementById('showAllBtn');

const showActiveBtn = document.getElementById('showActiveBtn');

const sortByNameBtn = document.getElementById('sortByNameBtn');

/\*\*

 \* Renders the employee list based on the provided array.

 \* @param {Array<Object>} employeeArray - The array of employee objects to display.

 \*/

function renderEmployees(employeeArray) {

    employeeListDiv.innerHTML = ''; // Clear previous content

    if (employeeArray.length === 0) {

        employeeListDiv.innerHTML = '<p>No employees to display.</p>';

        return;

    }

    employeeArray.forEach(employee => {

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

        employeeCard.classList.add('employee-card');

        employeeCard.dataset.employeeId = employee.id; // Store ID for toggling

        const statusClass = employee.isActive ? 'active' : 'inactive';

        const statusText = employee.isActive ? 'Active' : 'Inactive';

        employeeCard.innerHTML = `

            <h3>${employee.name}</h3>

            <p><strong>Position:</strong> ${employee.position}</p>

            <p><strong>Department:</strong> ${employee.department}</p>

            <p><strong>Status:</strong> <span class="status ${statusClass}">${statusText}</span></p>

            <button class="toggle-status-btn" data-employee-id="${employee.id}">Toggle Status</button>

        `;

        employeeListDiv.appendChild(employeeCard);

    });

    // Attach event listeners to new toggle buttons

    attachToggleStatusListeners();

}

/\*\*

 \* Attaches event listeners to all 'Toggle Status' buttons.

 \*/

function attachToggleStatusListeners() {

    document.querySelectorAll('.toggle-status-btn').forEach(button => {

        // Remove existing listener to prevent duplicates if re-rendering

        button.removeEventListener('click', toggleEmployeeStatus);

        button.addEventListener('click', toggleEmployeeStatus);

    });

}

/\*\*

 \* Toggles an employee's active status and updates the DOM.

 \* @param {Event} event - The click event.

 \*/

function toggleEmployeeStatus(event) {

    const employeeId = parseInt(event.target.dataset.employeeId);

    const employeeIndex = employees.findIndex(emp => emp.id === employeeId);

    if (employeeIndex > -1) {

        // Toggle the isActive property

        employees[employeeIndex].isActive = !employees[employeeIndex].isActive;

        // Re-render the list to reflect the change

        // We re-render the currently displayed view (e.g., all, or filtered active)

        // This makes the UI consistent with the applied filter/sort

        let currentEmployeesToRender = [...employees]; // Start with all employees

        // Check if the "Show Only Active Employees" button is currently selected/active

        // This is a simple way; for complex UIs, use state management

        if (showActiveBtn.classList.contains('active-filter')) { // Assuming we add an 'active-filter' class

             currentEmployeesToRender = employees.filter(emp => emp.isActive);

        } else if (sortByNameBtn.classList.contains('active-sort')) { // Assuming 'active-sort' class

             currentEmployeesToRender.sort((a, b) => a.name.localeCompare(b.name));

        }

        renderEmployees(currentEmployeesToRender);

        // Optionally, update the specific card directly without re-rendering the whole list

        // const card = event.target.closest('.employee-card');

        // const statusSpan = card.querySelector('.status');

        // if (employees[employeeIndex].isActive) {

        //     statusSpan.textContent = 'Active';

        //     statusSpan.classList.remove('inactive');

        //     statusSpan.classList.add('active');

        // } else {

        //     statusSpan.textContent = 'Inactive';

        //     statusSpan.classList.remove('active');

        //     statusSpan.classList.add('inactive');

        // }

    }

}

// Initial display of all employees when the page loads

renderEmployees(employees);

// --- Higher-Order Functions and Button Event Listeners ---

// Filter Active Employees

showActiveBtn.addEventListener('click', () => {

    const activeEmployees = employees.filter(employee => employee.isActive);

    renderEmployees(activeEmployees);

    // Add/remove classes to indicate active filter/sort for better UX

    showActiveBtn.classList.add('active-filter');

    showAllBtn.classList.remove('active-filter');

    sortByNameBtn.classList.remove('active-sort');

});

// Show All Employees

showAllBtn.addEventListener('click', () => {

    renderEmployees(employees);

    showAllBtn.classList.add('active-filter');

    showActiveBtn.classList.remove('active-filter');

    sortByNameBtn.classList.remove('active-sort');

});

// Sort Employees by Name

sortByNameBtn.addEventListener('click', () => {

    // Create a copy to avoid modifying the original 'employees' array directly if not desired

    // Or, if sorting the original is okay, just use employees.sort()

    const sortedEmployees = [...employees].sort((a, b) => a.name.localeCompare(b.name));

    renderEmployees(sortedEmployees);

    sortByNameBtn.classList.add('active-sort');

    showAllBtn.classList.remove('active-filter');

    showActiveBtn.classList.remove('active-filter');

});

// Map Employee Names to a New Array and display in console

const employeeNames = employees.map(employee => employee.name);

console.log('All Employee Names (using map()):', employeeNames);

// Optional: Add some basic styling for active/selected buttons

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

style.textContent = `

    button.active-filter, button.active-sort {

        background-color: #28a745; /\* A success green \*/

        color: white;

        border-color: #28a745;

    }

    button.active-filter:hover, button.active-sort:hover {

        background-color: #218838;

    }

`;

document.head.appendChild(style);

README :

**# LaunchDarkly Employee Portal**

This project implements a dynamic employee portal demonstrating fundamental JavaScript concepts: arrays, objects, DOM manipulation, higher-order functions (`filter`, `sort`, `map`), and event handling.

**## Project Structure**

\* `index.html`: The main HTML file providing the structure for the employee display and control buttons.

\* `employees.js`: The JavaScript file containing all the logic for managing employee data, rendering, filtering, sorting, and handling user interactions.

**## How to Run**

1.  Save `index.html` and `employees.js` in the same directory.

2.  Open `index.html` in your web browser. (You can simply double-click the file, or use an extension like Live Server in VS Code for a better development experience).

3.  Open your browser's developer console (usually F12) to see the output from the `map()` function.

**## Assignment Tasks & Implementation Details**

**### 1. Create and Display Employee Data (Arrays and Objects)**

\* **\*\*Data Structure\*\***: Employee data is stored in a JavaScript array named `employees`. Each element in this array is an `object` representing an employee, with properties like `id`, `name`, `position`, `department`, and `isActive`.

    ```javascript

    let employees = [

        { id: 1, name: 'Alice Johnson', position: 'Software Engineer', department: 'Engineering', isActive: true },

        // ... more employee objects

    ];

    ```

\* **\*\*Dynamic Display (DOM Manipulation)\*\***:

    \* The `renderEmployees(employeeArray)` function is responsible for dynamically generating the HTML.

    \* It clears the existing content of the `<div id="employee-list">`.

    \* It then iterates through the provided `employeeArray` using `forEach()`.

    \* For each employee, a new `<div>` element with the class `employee-card` is created.

    \* Employee details (name, position, department, and status) are injected into the `innerHTML` of this `div`.

    \* The status text and styling (`active` or `inactive` class) are dynamically set based on the `isActive` property.

    \* Each generated `employee-card` is appended to the `employee-list` container in the HTML.

**### 2. Filter Active Employees (Higher-Order Function - `filter()`)**

\* The `filter()` higher-order function is used on the `employees` array to create a new array containing only active employees.

\* The `showActiveBtn` has an event listener. When clicked, it calls:

    ```javascript

    const activeEmployees = employees.filter(employee => employee.isActive);

    renderEmployees(activeEmployees);

    ```

    This filters the `employees` array, keeping only those objects where `employee.isActive` is `true`, and then re-renders the display with this filtered subset.

**### 3. Sort Employees by Name (Higher-Order Function - `sort()`)**

\* The `sort()` higher-order function is used to arrange the employees alphabetically by their `name`.

\* The `sortByNameBtn` has an event listener. When clicked, it calls:

    ```javascript

    const sortedEmployees = [...employees].sort((a, b) => a.name.localeCompare(b.name));

    renderEmployees(sortedEmployees);

    ```

    \* `[...employees]` creates a shallow copy of the `employees` array to avoid modifying the original array's order directly if it's not the desired behavior for subsequent operations (though in this case, it simply ensures the `employees` array maintains its original order if you were to re-render "all" employees without another sort).

    \* The `sort()` method uses a compare function (`(a, b) => a.name.localeCompare(b.name)`) to sort strings alphabetically.

    \* The `renderEmployees` function is then called with the newly sorted array.

**### 4. Map Employee Names to a New Array (Higher-Order Function - `map()`)**

\* The `map()` higher-order function transforms the `employees` array into a new array containing only the names of the employees.

\* This operation is performed once when the script loads:

    ```javascript

    const employeeNames = employees.map(employee => employee.name);

    console.log('All Employee Names (using map()):', employeeNames);

    ```

\* The resulting `employeeNames` array is then logged to the browser's console.

**### 5. Bonus: Update Employee Status (Basic DOM Manipulation with Events)**

\* **\*\*Toggle Button\*\***: A "Toggle Status" button is dynamically added next to each employee's status within their respective employee card.

\* **\*\*Event Listener\*\***: The `attachToggleStatusListeners()` function is called after each `renderEmployees` operation to ensure all newly created toggle buttons have an event listener attached. This listener calls the `toggleEmployeeStatus` function when a button is clicked.

\* **\*\*`toggleEmployeeStatus(event)` Function\*\***:

    \* It identifies the clicked employee's `id` using `event.target.dataset.employeeId`.

    \* It uses `findIndex()` to locate the corresponding employee object in the `employees` array.

    \* It then flips the `isActive` boolean property (`employees[employeeIndex].isActive = !employees[employeeIndex].isActive;`).

    \* Finally, it re-renders the entire employee list (`renderEmployees(...)`) to reflect the updated status in the UI. This ensures that any active filters or sorts are reapplied correctly.

**## Challenges Faced & Overcoming Them**

1.  **\*\*Re-attaching Event Listeners\*\***:

    \* **\*\*Challenge\*\***: When dynamically re-rendering the entire employee list (e.g., after filtering or sorting), any previously attached event listeners on the `Toggle Status` buttons would be lost because the old elements are removed from the DOM and new ones are created. This meant the toggle buttons wouldn't work after the first rendering or any filter/sort action.

    \* **\*\*Overcoming\*\***: The `attachToggleStatusListeners()` function was created. It's called *\*after\** every `renderEmployees()` call. Inside this function, `querySelectorAll('.toggle-status-btn')` selects all existing toggle buttons, and an event listener is attached to each. `removeEventListener` is also used defensively to prevent duplicate listeners if `renderEmployees` is called multiple times without fully clearing the DOM.

2.  **\*\*Maintaining Current View State After Status Toggle\*\***:

    \* **\*\*Challenge\*\***: When an employee's status is toggled, if the user was currently viewing only "Active Employees" or a "Sorted by Name" list, simply toggling the `isActive` property in the `employees` array and re-rendering with `employees` would revert the view to "all unsorted employees".

    \* **\*\*Overcoming\*\***: In `toggleEmployeeStatus`, after updating the `isActive` property, I added logic to check which filter/sort button is "active" (by adding a CSS class like `active-filter` or `active-sort` to the buttons). Based on this, the appropriate filter or sort is re-applied to the `employees` array before calling `renderEmployees`, ensuring the UI consistently reflects the user's current view preference.

3.  **\*\*Preventing Direct Modification of Original Array for Sort\*\***:

    \* **\*\*Challenge\*\***: The `sort()` method modifies the array in place. If we wanted to preserve the original `employees` array's order for a "Show All" function, directly sorting it would be problematic.

    \* **\*\*Overcoming\*\***: For the `sortByNameBtn` click, I used the spread syntax (`[...employees]`) to create a shallow copy of the `employees` array before sorting it. This way, the original `employees` array remains in its initial order, while the `renderEmployees` function receives a sorted copy.

This project effectively demonstrates the power of JavaScript in creating interactive and dynamic web applications by leveraging fundamental data structures and functional programming paradigms.

SCREENSHOT:

