USER MENU PERMISSIONS

Sure, let's break this down and create a solution for your use case where you can toggle menu options to "cross" them out or disable them for a specific user.

**Objective**

1. **Extract the <ul><li> items** from the template literal and represent them in an array.
2. **Group the menu options** by their parent <li>.
3. Create a mechanism to **toggle the selection** of specific menu items (cross them out) based on user permissions.
4. Store the **selected menu items** to track which options are disabled.

**Approach**

1. **Extract menu items**:  
   We'll loop through the <ul><li> structure, collect the items, and organize them into a structured object, grouped by their parent categories (e.g., "Sales", "Purchases").
2. **Toggle menu item**:  
   We'll add a mechanism to toggle the "crossed-out" state for each item. This could be done via a simple checkbox or toggle button next to each item.
3. **Store the crossed-out items**:  
   We'll store the "disabled" state (crossed-out or not) of each menu item in an object, using the menu-ref or class as a unique identifier.

**Code Breakdown:**

1. **Extract menu items and group them:**

We can use JavaScript to parse the HTML string and create a structured array of objects for each menu category.

const menuItems = `

<li class="sales"><i class="fa fa-dollar-sign"></i> Sales

<ul class="dropdown submenu">

<li menu-ref="A01" class="SalesInvoice data-entry"><i class="fa fa-th-list"></i> Sales Invoice</li>

<hr class="menuLine">

<li menu-ref="A02" class="salesRankingByLocation">Sales Ranking by Location</li>

<li menu-ref="A03" class="salesRankingByBrand">Sales Ranking by Brand</li>

<li menu-ref="A04" id="salesRankingByCategory">Sales Ranking by Category</li>

<li menu-ref="A05" id="dailySalesSum">Daily Sales Summary</li>

<hr class="menuLine">

<li menu-ref="A06" >Sales By SKU</li>

</ul>

</li>

<!-- Other menu categories... -->

`;

function parseMenuItems(menuHTML) {

const parser = new DOMParser();

const doc = parser.parseFromString(menuHTML, 'text/html');

const menuList = Array.from(doc.querySelectorAll('li')).map(parent => {

const category = parent.className;

const subMenu = Array.from(parent.querySelectorAll('.submenu li')).map(item => {

return {

name: item.textContent.trim(),

ref: item.getAttribute('menu-ref'),

id: item.id || null,

className: item.className,

selected: false // Initially, no items are selected (not crossed-out)

};

});

return { category, items: subMenu };

});

return menuList;

}

const menu = parseMenuItems(menuItems);

console.log(menu);

The menu will now be structured like this:

[

{

category: 'sales',

items: [

{ name: 'Sales Invoice', ref: 'A01', selected: false, className: 'SalesInvoice data-entry' },

{ name: 'Sales Ranking by Location', ref: 'A02', selected: false, className: 'salesRankingByLocation' },

// More items...

]

},

// Other categories...

]

1. **Toggling menu items (crossing out):**

Now, let’s add the ability to toggle the selected state for each menu item, which can be visualized as crossing out items.

function toggleMenuItem(ref) {

// Find the item in the menu and toggle its 'selected' state

menu.forEach(category => {

category.items.forEach(item => {

if (item.ref === ref) {

item.selected = !item.selected;

}

});

});

}

// Example of toggling an item:

toggleMenuItem('A01'); // Toggle "Sales Invoice"

toggleMenuItem('B02'); // Toggle "Stock Receiving Form"

1. **Storing selected items:**

For storing selected items (e.g., crossing out for a user), you could maintain a list or object to store which items are disabled for a particular user:

const userPermissions = {

"John Smith": []

};

function updateUserPermissions(user, menuItemRef) {

const userPerms = userPermissions[user] || [];

const itemIndex = userPerms.indexOf(menuItemRef);

if (itemIndex === -1) {

userPerms.push(menuItemRef); // Add to the disabled list

} else {

userPerms.splice(itemIndex, 1); // Remove from the disabled list

}

userPermissions[user] = userPerms;

}

// Example: John Smith shouldn't have access to "Sales Invoice" and "Stock Receiving Form"

updateUserPermissions('John Smith', 'A01');

updateUserPermissions('John Smith', 'B02');

console.log(userPermissions);

**Putting It All Together:**

// Sample menu structure parsing

const menu = parseMenuItems(menuItems);

// User Permissions

const userPermissions = {

"John Smith": ['A01', 'B02'], // Example: John Smith cannot access these

};

// Render function (can use to update the DOM)

function renderMenu() {

const menuContainer = document.getElementById('menu-container');

menuContainer.innerHTML = '';

menu.forEach(category => {

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

categoryElement.innerHTML = `<h3>${category.category}</h3>`;

const submenu = document.createElement('ul');

category.items.forEach(item => {

const menuItem = document.createElement('li');

const isSelected = userPermissions['John Smith'].includes(item.ref);

menuItem.classList.toggle('disabled', isSelected);

menuItem.innerHTML = `

<input type="checkbox" ${isSelected ? 'checked' : ''}

onclick="toggleMenuItem('${item.ref}'); updateUserPermissions('John Smith', '${item.ref}')">

${item.name}

`;

submenu.appendChild(menuItem);

});

categoryElement.appendChild(submenu);

menuContainer.appendChild(categoryElement);

});

}

// Call render

renderMenu();

**HTML Structure**

<div id="menu-container"></div>

**Final Thoughts:**

* **Data Structure**: The data for menu items and user permissions is stored in JavaScript objects/arrays. Each menu item is identified by menu-ref, which helps track user selections.
* **UI Interaction**: Toggling a checkbox will update the item’s state (crossed-out or available).
* **User-Specific Permissions**: By using a simple map (userPermissions), you can specify which items are disabled for each user.

This setup can be dynamically updated, and the UI can be modified as per requirements. You could integrate this solution with your back-end to store user preferences persistently.

RUNNING TIMER

<div id="loadingIndicator" style="display:none;">

<i class="fa fa-spinner fa-spin"></i> Loading data... Please wait

<div id="runningTime"></div>

</div>

<script>

let timerInterval;

let elapsedTime = 0;

function startTimer() {

timerInterval = setInterval(() => {

elapsedTime++;

const minutes = Math.floor(elapsedTime / 60);

const seconds = elapsedTime % 60;

document.getElementById('runningTime').textContent =

`Elapsed Time: ${minutes}m ${seconds}s`;

}, 1000);

}

document.getElementById('loadingIndicator').style.display = 'flex';

startTimer();

try {

// Build query parameters

const url = new URL('http://localhost:3000/sales/SalesCompStore');

const params = new URLSearchParams();

if (cBrandNum) params.append('BrandNum', cBrandNum);

// Send request with query parameters

const response = await fetch(`${url}?${params.toString()}`);

if (!response.ok) {

throw new Error('Network response was not ok');

}

const listCounter = document.getElementById('saleRank1Counter');

const data = await response.json();

listCounter.innerHTML = `${data.length} Records`;

showNotification(`${data.length} Records fetched`);

// Clear the timer once data is fetched

clearInterval(timerInterval);

} catch (error) {

console.error('Fetch error:', error);

displayErrorMsg(error, 'Fetch error');

} finally {

// Hide loading indicator and clear timer in case of an error

document.getElementById('loadingIndicator').style.display = 'none';

clearInterval(timerInterval); // Ensure timer is cleared even on error

}

</script>