**/\* Group tree view \*/**

**Html code :-**

<tree-root

#tree

class="expand-tree"

[nodes]="treeDataSource"

[options]="options"

(updateData)="onUpdateData(tree, $event)"

(activate)="onActivateNode($event, tree)"

(initialized)="initTree($event)"

[state]="state"

>

<ng-template #treeNodeTemplate let-node let-index="index">

<mat-icon \*ngIf="node.data?.type === 'Building'">location\_city</mat-icon>

<mat-icon \*ngIf="node.data?.type === 'Floor'" [contextMenu]="basicMenu">layers</mat-icon>

<mat-icon \*ngIf="node.data?.type === 'Zone'" [contextMenu]="basicMenu">view\_quilt</mat-icon>

<mat-icon \*ngIf="node.data?.type === 'Group'" [contextMenu]="basicMenu">grain</mat-icon>

<mat-icon \*ngIf="node.data?.type === 'DataPoints'" [contextMenu]="basicMenu">offline\_bolt</mat-icon>

<span

\*ngIf="node.data?.isEdit === undefined || !node.data?.isEdit"

[contextMenu]="basicMenu"

[contextMenuSubject]="node.data"

>

{{ node.data.name ? node.data.name : node.data.tagName }}

</span>

<div class="editing-dp" \*ngIf="node.data?.isEdit">

<mat-form-field>

<input

matInput

type="text"

maxlength="30"

[(ngModel)]="newValue"

value="{{ node.data.name }}"

class="nodetext"

(keydown.enter)="updateNode(node)"

/>

<mat-icon

matSuffix

class="icon-color-tree edit-icon-tree"

(click)="updateNode(node)"

>check\_box</mat-icon

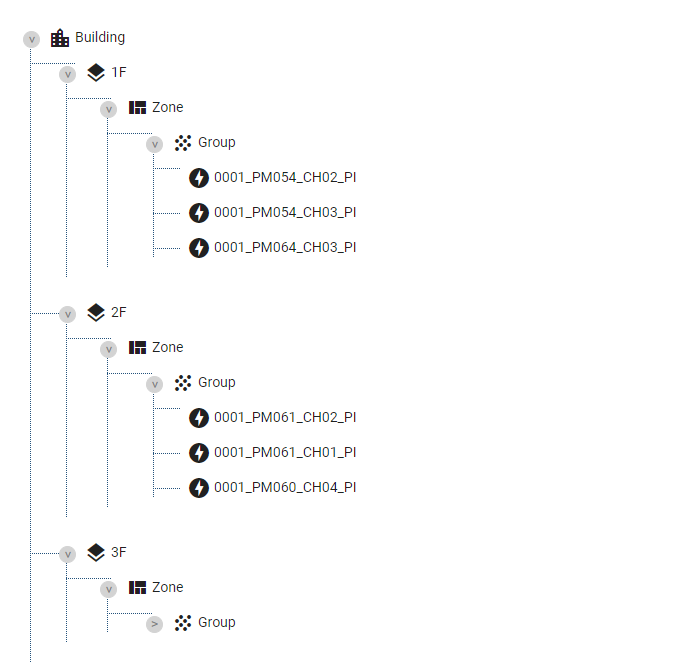
>

</mat-form-field>

</div>

</ng-template>

</tree-root>



**/\* Group .ts file \*/**

actionMapping: IActionMapping = {

mouse: {

contextMenu: (tree, node, $event) => {

// if (node.hasChildren) TREE\_ACTIONS.TOGGLE\_EXPANDED(tree, node, $event);

},

click: (tree, node, $event) => {

// condition for user click on datapoint of diiffret group

// Then set active to true and cleared selectedGroupIdForUnassign array

if (

node &&

node.parent &&

node.parent.data &&

this.selectedGroupIdForUnassign === undefined

) {

this.selectedGroupIdForUnassign = node.parent.data.\_id;

}

if (

node &&

node.parent &&

node.parent.data &&

this.selectedGroupIdForUnassign &&

this.selectedGroupIdForUnassign !== node.parent.data.\_id

) {

this.selectedItemForUnassign = [];

node.setIsActive(true);

this.selectedGroupIdForUnassign = node.parent.data.\_id;

if (node.data.\_id === '') {

TREE\_ACTIONS.TOGGLE\_ACTIVE(tree, node, $event);

}

return;

}

this.activeSingleAndMultiSelection(node, tree, $event);

// if (node.hasChildren) TREE\_ACTIONS.TOGGLE\_EXPANDED(tree, node, $event);

},

drop: (tree, node, $event, { from, to }) => {

this.groupingService.activeNode = node.data.\_id;

this.isAssignedEnabled = true;

let datapoints = [];

if (from.length > 0) {

from.forEach((element) => {

datapoints.push(element.\_id);

});

} else {

datapoints.push(from.\_id);

}

if (to.parent.data.type !== 'Group') {

this.notificationService.showWarning(

this.authService.languageObject.grouping.warningMessage.dataPointAssignment,

this.authService.languageObject.grouping.wrongAssignment

);

} else if (to.parent.data.type === 'Group' && datapoints.length > 0) {

// Need to check already assigned DP

const existingDatapointIds = (to.parent.data?.children || []).map(d => d.\_id);

if (existingDatapointIds.length >= this.maxDatapointsAssignementLimit) {

this.notificationService.showWarning(

this.authService.languageObject.grouping.warningMessage.maxDatapointsReached.replace('COUNT', this.maxDatapointsAssignementLimit),

this.authService.languageObject.grouping.wrongAssignment

);

return;

}

const maxDatapointsCanAssign = this.maxDatapointsAssignementLimit - existingDatapointIds.length;

// Move unassigned datapoints to top

const alreadyAssignedDP = datapoints.filter(\_id => existingDatapointIds.indexOf(\_id) > -1);

const notAssignedDP = datapoints.filter(\_id => existingDatapointIds.indexOf(\_id) == -1);

const sortedDatapointIds = notAssignedDP.concat(alreadyAssignedDP);

const dataPointIds = sortedDatapointIds.slice(0, maxDatapointsCanAssign);

const skippedDatapointsCount = sortedDatapointIds.length - dataPointIds.length;

if (skippedDatapointsCount > 0) {

// Show skip message due to max limit issue.

this.notificationService.showWarning(

this.authService.languageObject.grouping.warningMessage.skippedDueToMaxLimit.replace('COUNT', skippedDatapointsCount).replace('DP\_COUNT', this.maxDatapointsAssignementLimit),

this.authService.languageObject.grouping.wrongAssignment

);

}

const data: AssignDatapoints = {

groupId: to.parent.data.\_id,

dataPointIds: dataPointIds

};

this.store.dispatch(new AssignDatapointsGrouping(data));

this.selectedItemForUnassign = [];

}

},

},

keys: {

[KEYS.ENTER]: (tree, node, $event) => {

// node.expandAll();

},

[KEYS.RIGHT]: (tree, node, $event) => {

// node.expandAll();

},

},

};

options: ITreeOptions = {

// displayField: 'nodeName',

isExpandedField: 'expanded',

idField: '\_id',

hasChildrenField: 'nodes',

actionMapping: this.actionMapping,

nodeHeight: 30,

allowDrag: (node) => {

return false;

},

allowDrop: (node, { parent, index }) => {

if (parent.data.type === 'Group') {

return true;

}

return false;

},

allowDragoverStyling: true,

levelPadding: 10,

useVirtualScroll: true,

animateExpand: true,

scrollOnActivate: true,

animateSpeed: 30,

animateAcceleration: 1.2,

scrollContainer: document.documentElement, // HTML

};