**import** {isOverDifferentColumn, getClassesForDropZone, getActionItem, defaultGetDeniedDropZoneProps} **from** '../helper';  
**import** s **from** '../../Column.mod.scss';  
  
describe('helper', () => {  
 describe('isOverDifferentColumn', () => {  
 test('when `sourceGroupId` is equal to `groupId`', () => {  
 **const** dragItem = {  
 sourceGroupId: 5,  
 };  
 **const** group = {  
 groupId: 5,  
 };  
 expect(isOverDifferentColumn(dragItem, group)).toBe(**false**);  
 });  
  
 test('when `sourceGroupId` is not equal to `groupId`', () => {  
 **const** dragItem = {  
 sourceGroupId: 5,  
 };  
 **const** group = {  
 groupId: 7,  
 };  
 expect(isOverDifferentColumn(dragItem, group)).toBe(**true**);  
 });  
 });  
  
 describe('getClassesForDropZone', () => {  
 test('when both `canDrop` and `isDragOver` are true', () => {  
 **const** props = {  
 canDrop: **true**,  
 isDragOver: **true**,  
 };  
 expect(getClassesForDropZone(props)).toBe('blueDropZone greenDropZone columnContentDropZoneAllowed');  
 });  
  
 test('when `canDrop` is true', () => {  
 **const** props = {  
 canDrop: **true**,  
 isDragOver: **false**,  
 };  
 expect(getClassesForDropZone(props)).toBe('blueDropZone');  
 });  
  
 test('when `isDragOver` is true', () => {  
 **const** props = {  
 canDrop: **false**,  
 isDragOver: **true**,  
 };  
 expect(getClassesForDropZone(props)).toBe('');  
 });  
  
 test('when both `canDrop` and `isDragOver` are false', () => {  
 **const** props = {  
 canDrop: **false**,  
 isDragOver: **false**,  
 };  
 expect(getClassesForDropZone(props)).toBe('');  
 });  
 });  
  
 describe('getActionItem', () => {  
 test('when shiftKey is pressed and user select only one item.', () => {  
 **const** itemInfo = {  
 id: 4,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 4: **false**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 },  
 data: {  
 items: [  
 {  
 id: 4,  
 },  
 {  
 id: 1,  
 },  
 ],  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 4: **true**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **true**,  
 metaKey: **false**,  
 ctrlKey: **false**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey is pressed and user select more than one items above the clickItem.', () => {  
 **const** itemInfo = {  
 id: 3,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 4: **false**,  
 1: **true**,  
 3: **false**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 1: 1,  
 3: 2,  
 },  
 data: {  
 items: [  
 {  
 id: 4,  
 },  
 {  
 id: 1,  
 },  
 {  
 id: 3,  
 },  
 ],  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 4: **false**,  
 1: **true**,  
 3: **true**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 1: 1,  
 3: 2,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **true**,  
 metaKey: **false**,  
 ctrlKey: **false**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey is pressed and user select more than one items below the clickItem.', () => {  
 **const** itemInfo = {  
 id: 4,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 4: **false**,  
 1: **true**,  
 3: **false**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 1: 1,  
 3: 2,  
 },  
 data: {  
 items: [  
 {  
 id: 4,  
 },  
 {  
 id: 1,  
 },  
 {  
 id: 3,  
 },  
 ],  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 4: **true**,  
 1: **true**,  
 3: **false**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 1: 1,  
 3: 2,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **true**,  
 metaKey: **false**,  
 ctrlKey: **false**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey is not pressed but metaKey or ctrlKey is pressed and item id of selected item is true', () => {  
 **const** itemInfo = {  
 id: 2,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 2: **true**,  
 3: **true**,  
 4: **false**,  
 },  
 itemIdToIndex: {  
 2: 0,  
 3: 1,  
 4: 3,  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 3: **true**,  
 4: **false**,  
 },  
 itemIdToIndex: {  
 3: 1,  
 4: 3,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **false**,  
 metaKey: **true**,  
 ctrlKey: **true**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey is not pressed but metaKey or ctrlKey is pressed and item id of selected item is false', () => {  
 **const** itemInfo = {  
 id: 4,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 2: **true**,  
 3: **true**,  
 4: **false**,  
 },  
 itemIdToIndex: {  
 2: 0,  
 3: 1,  
 4: 2,  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 2: **true**,  
 3: **true**,  
 4: **true**,  
 },  
 itemIdToIndex: {  
 2: 0,  
 4: 5,  
 3: 1,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **false**,  
 metaKey: **true**,  
 ctrlKey: **true**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey, metaKey and ctrlKey are not selected and item id of selected item is false', () => {  
 **const** itemInfo = {  
 id: 4,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 2: **true**,  
 3: **true**,  
 4: **true**,  
 },  
 itemIdToIndex: {  
 2: 0,  
 3: 1,  
 4: 2,  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {  
 4: **true**,  
 },  
 itemIdToIndex: {  
 4: 5,  
 },  
 },  
 };  
 **const** e = {  
 shiftKey: **false**,  
 metaKey: **false**,  
 ctrlKey: **false**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
  
 test('when shiftKey, metaKey and ctrlKey are not selected and item id of selected item is true', () => {  
 **const** itemInfo = {  
 id: 4,  
 itemIndex: 5,  
 };  
 **const** props = {  
 selectedItemIds: {  
 4: **true**,  
 },  
 itemIdToIndex: {  
 4: 0,  
 },  
 };  
 **const** output = {  
 type: 'ON\_ITEM\_SELECT',  
 payload: {  
 itemIdsMap: {},  
 itemIdToIndex: {},  
 },  
 };  
 **const** e = {  
 shiftKey: **false**,  
 metaKey: **false**,  
 ctrlKey: **false**,  
 };  
 **const** getItemId = **function** (item) {  
 **return** item.id;  
 };  
 expect(getActionItem(itemInfo, e, props, getItemId)).toEqual(output);  
 });  
 });  
  
 describe('defaultGetDeniedDropZoneProps', () => {  
 test('Just check function without parameters', () => {  
 **const** obj = {  
 icon: 'denied',  
 iconClassName: `${s.deniedDropZoneIcon} fill-ash`,  
 primaryText: \_\_('You cannot do this operation.'),  
 secondaryText: \_\_('Check your permissions!'),  
 };  
 expect(defaultGetDeniedDropZoneProps()).toEqual(obj);  
 });  
 });  
});