i have an ag grid in react typescript and dont want to use the built in functionality for selecting which columns are hidden and whcih arent. I want to click an icon button and have a context menu come up with the list of all possible columns with checkboxes beside them, then when i click apply in that cntext menu, the context menu closes and the columns are shown as selected

ChatGPT

To achieve the functionality you described, you'll need to create a custom context menu with checkboxes for each column in your AG Grid component in React with TypeScript. Here's a step-by-step guide on how to implement this:

Set Up AG Grid and Data:

Make sure you have AG Grid and your data properly set up in your React component.

Create State for Column Visibility:

Create a state to manage the visibility of columns. You can use the useState hook to achieve this. Initialize the state with the default visibility settings.

tsx

Copy code

import React, { useState } from 'react';

const MyGridComponent: React.FC = () => {

const [columnVisibility, setColumnVisibility] = useState<{ [key: string]: boolean }>({

column1: true,

column2: true,

// ... other columns

});

// AG Grid configuration and data binding

// ...

};

Create Context Menu Component:

Create a separate component for the context menu with checkboxes for each column. This component will receive the columnVisibility state and the setColumnVisibility function as props.

tsx

Copy code

import React from 'react';

interface ContextMenuProps {

columnVisibility: { [key: string]: boolean };

setColumnVisibility: (newVisibility: { [key: string]: boolean }) => void;

}

const ContextMenu: React.FC<ContextMenuProps> = ({ columnVisibility, setColumnVisibility }) => {

const handleCheckboxChange = (columnName: string) => {

setColumnVisibility((prevVisibility) => ({

...prevVisibility,

[columnName]: !prevVisibility[columnName],

}));

};

return (

<div className="context-menu">

{Object.keys(columnVisibility).map((columnName) => (

<label key={columnName}>

<input

type="checkbox"

checked={columnVisibility[columnName]}

onChange={() => handleCheckboxChange(columnName)}

/>

{columnName}

</label>

))}

<button onClick={() => /\* Apply changes and close context menu \*/}>Apply</button>

</div>

);

};

export default ContextMenu;

Integrate Context Menu:

In your main component, render the AG Grid and the custom context menu. When the user clicks the icon button, you can toggle the visibility of the context menu.

tsx

Copy code

const MyGridComponent: React.FC = () => {

const [showContextMenu, setShowContextMenu] = useState(false);

// Toggle context menu visibility

const toggleContextMenu = () => {

setShowContextMenu(!showContextMenu);

};

// Render AG Grid and context menu

return (

<div>

<button onClick={toggleContextMenu}>Show Context Menu</button>

{showContextMenu && (

<ContextMenu columnVisibility={columnVisibility} setColumnVisibility={setColumnVisibility} />

)}

{/\* Render AG Grid here \*/}

</div>

);

};

Apply Column Visibility Changes:

In the ContextMenu component, implement the logic for applying the column visibility changes and closing the context menu when the "Apply" button is clicked.

Update AG Grid Configuration:

Finally, in your AG Grid configuration, use the columnVisibility state to determine which columns should be shown or hidden based on the user's selections.

By following these steps, you can create a custom context menu with checkboxes for each column in your AG Grid component in React with TypeScript. This approach gives you full control over the visibility of columns and provides a more customized user experience.

User

does the ag grid have to be rendered in the same div as the context menu

ChatGPT

No, the AG Grid and the custom context menu don't have to be rendered in the same div. They can be separate components and rendered independently. The key is to manage the state that holds the visibility of columns (columnVisibility) in a common parent component so that both the AG Grid and the context menu components can access and update this state.

Here's how you could structure your components without rendering the AG Grid and the context menu in the same div:

App Component (Parent Component):

tsx

Copy code

import React, { useState } from 'react';

import MyGridComponent from './MyGridComponent';

import ContextMenu from './ContextMenu';

const App: React.FC = () => {

const [showContextMenu, setShowContextMenu] = useState(false);

const [columnVisibility, setColumnVisibility] = useState<{ [key: string]: boolean }>({

column1: true,

column2: true,

// ... other columns

});

const toggleContextMenu = () => {

setShowContextMenu(!showContextMenu);

};

return (

<div>

<button onClick={toggleContextMenu}>Show Context Menu</button>

{showContextMenu && (

<ContextMenu

columnVisibility={columnVisibility}

setColumnVisibility={setColumnVisibility}

/>

)}

<MyGridComponent columnVisibility={columnVisibility} />

</div>

);

};

export default App;

MyGridComponent:

tsx

Copy code

import React from 'react';

import { AgGridReact } from 'ag-grid-react';

const MyGridComponent: React.FC<{ columnVisibility: { [key: string]: boolean } }> = ({

columnVisibility,

}) => {

// AG Grid configuration using columnVisibility for column visibility settings

// ...

return <AgGridReact /\* AG Grid configuration \*/ />;

};

export default MyGridComponent;

ContextMenu:

tsx

Copy code

import React from 'react';

interface ContextMenuProps {

columnVisibility: { [key: string]: boolean };

setColumnVisibility: (newVisibility: { [key: string]: boolean }) => void;

}

const ContextMenu: React.FC<ContextMenuProps> = ({ columnVisibility, setColumnVisibility }) => {

const handleCheckboxChange = (columnName: string) => {

setColumnVisibility((prevVisibility) => ({

...prevVisibility,

[columnName]: !prevVisibility[columnName],

}));

};

return (

<div className="context-menu">

{Object.keys(columnVisibility).map((columnName) => (

<label key={columnName}>

<input

type="checkbox"

checked={columnVisibility[columnName]}

onChange={() => handleCheckboxChange(columnName)}

/>

{columnName}

</label>

))}

<button onClick={() => /\* Apply changes and close context menu \*/}>Apply</button>

</div>

);

};

export default ContextMenu;

In this structure, the AG Grid and the context menu are both rendered in the App component's JSX, but they are independent components and don't need to be in the same div. The columnVisibility state is managed by the App component and is passed down to the MyGridComponent and ContextMenu components as needed. This way, you can achieve the desired behavior of showing the context menu with checkboxes for column visibility and applying the changes to the AG Grid columns.

User

great

ChatGPT

I'm glad I could help! If you have any more questions or need further assistance, feel free to ask. Happy coding!

User

i also want to replace the inbuilt row grouping functionality, where you can group by column name and a chip appears at the top of the grid, and if you do multiple columns, you can drag the column chips around the top bar to rearrange the order of row grouping, how do i do this

ChatGPT

Replacing the built-in row grouping functionality with a custom implementation involves several steps. You'll need to handle grouping logic, UI elements like chips, and user interactions for reordering. Below, I'll outline the general approach to achieve this:

Custom Grouping Logic:

Implement your custom grouping logic by modifying your data source to reflect grouped rows. You would need to transform your data to create a hierarchical structure based on the selected group columns.

Render Chips for Grouping:

Render chips for the selected group columns. Each chip should represent a grouped column. You can use a list to display these chips at the top of the grid.

Drag and Drop for Reordering:

To enable the user to rearrange the order of grouping columns, you'll need to implement drag-and-drop functionality for the chips. React DnD (Drag and Drop) library is commonly used for such interactions.

Update Grouping and Data:

When the user rearranges the chips or changes grouping, you should update the grouping configuration and data accordingly. Your custom logic should handle regrouping the data based on the new grouping configuration.

Here's a high-level outline of how you can approach this:

tsx

Copy code

import React, { useState } from 'react';

import { useDrag, useDrop } from 'react-dnd';

// Define the types for dragging and dropping

const CHIP = 'chip';

// Chip component for grouping

const GroupChip = ({ column, index, moveChip }) => {

const [, ref] = useDrag({

type: CHIP,

item: { index },

});

const [, drop] = useDrop({

accept: CHIP,

hover: (draggedItem) => {

if (draggedItem.index !== index) {

moveChip(draggedItem.index, index);

draggedItem.index = index;

}

},

});

return (

<div ref={(node) => ref(drop(node))} className="group-chip">

{column}

</div>

);

};

// App component

const App = () => {

const [groupColumns, setGroupColumns] = useState<string[]>([]);

const [data, setData] = useState(/\* Your transformed grouped data \*/);

const handleColumnClick = (column) => {

if (groupColumns.includes(column)) {

setGroupColumns(groupColumns.filter((col) => col !== column));

} else {

setGroupColumns([...groupColumns, column]);

}

};

const handleChipMove = (fromIndex, toIndex) => {

const updatedColumns = [...groupColumns];

const [movedChip] = updatedColumns.splice(fromIndex, 1);

updatedColumns.splice(toIndex, 0, movedChip);

setGroupColumns(updatedColumns);

};

// Update data and grouping configuration whenever groupColumns change

// ...

return (

<div>

<div className="group-chips">

{groupColumns.map((column, index) => (

<GroupChip key={column} column={column} index={index} moveChip={handleChipMove} />

))}

</div>

{/\* Render AG Grid with custom data \*/}

</div>

);

};

Please note that this is a simplified example, and you'll need to integrate this logic with your AG Grid and ensure that the grouping and data transformation work according to your needs. The React DnD library can be used to manage the drag-and-drop interactions for the chips.

Remember that implementing custom grouping can be quite complex, and it requires a good understanding of both your data structure and the AG Grid API. Be sure to refer to the AG Grid documentation and customize the approach to suit your specific use case.