dnd-kit Implementation Guide

This document explains how **@dnd-kit** is used in this project to enable drag-and-drop functionality for scripts and folders.

Table of Contents

What is dnd-kit?

@dnd-kit is a modern, lightweight, and performant drag-and-drop toolkit for React. It provides:

- 🎯 **Flexible collision detection algorithms**
- 🔄 **Sortable lists** with automatic reordering
- 📦 **Droppable zones** for cross-list operations
- 🎨 **Customizable drag overlays**
- ♿ **Accessibility** support out of the box

Core Concepts

1. **DndContext**

The root component that manages the drag-and-drop state.

```
<DndContext
    sensors={sensors}
    collisionDetection={customCollisionDetection}
    onDragStart={handleDragStart}
    onDragEnd={handleDragEnd}
>
    {/* Draggable content here */}
</DndContext>
```

Key Props:

- `sensors`: Input methods (mouse, touch, keyboard)
- `collisionDetection`: Algorithm to detect overlaps
- `onDragStart`: Callback when drag begins
- `onDragEnd`: Callback when item is dropped

---

2. **SortableContext**

Enables items within a list to be reordered.

```
<SortableContext
    items={items.map(item => item.id)}
    strategy={verticalListSortingStrategy}
>
    {items.map(item => (
        <SortableItem key={item.id} item={item} />
    ))}
</SortableContext>
```

Key Props:

- `items`: Array of item IDs
- `strategy`: Sorting behavior (vertical, horizontal, grid)

---

3. **useSortable Hook**

Makes an individual item sortable (can drag AND be a drop target).

```
const {
    attributes, // Props for the draggable element
    listeners, // Event handlers for dragging
    setNodeRef, // Ref to attach to the DOM element
    transform, // Current position transformation
    transition, // CSS transition
    isDragging, // Boolean indicating drag state
} = useSortable({
    id: item.id,
    data: { type: "item", item }, // Custom metadata
});
```

---

4. **useDroppable Hook**

Creates a drop zone that can receive dragged items (cannot be dragged itself).

```
const {
    setNodeRef, // Ref for the droppable area
    isOver, // Boolean: true when item hovers over this zone
} = useDroppable({
    id: "drop-zone-1",
    data: { type: "folder", folderId: 123 },
});
```

5. **DragOverlay**

Renders a clone that follows the cursor while dragging (creates smooth animations).

```
<DragOverlay>
    {activeId ? (
        <div className="opacity-80">
            <ItemPreview item={activeItem} />
        </div>
    ) : null}
</DragOverlay>
```

Project Implementation

Architecture Overview

```
ScriptsColumn (DndContext)
    ├── SortableSubfolders
    │   └── CollapsableFolder (useSortable + useDroppable)
    │       - Can be dragged (for reordering)
    │       - Can accept scripts (as drop target)
    │
    └── SortableScripts
        └── SortableScriptItem (useSortable)
            - Can be dragged into folders or reordered
```

---

1. Main DndContext Setup

File: `src/app-component/ScriptsColumn/ScriptsColumn.tsx`

```
export default function ScriptsColumn() {
    const [activeId, setActiveId] = useState<number | null>(null);
    const [activeType, setActiveType] = useState<"script" | "folder" |
null>(null);

    const sensors = useSensors(
        useSensor(PointerSensor),
        useSensor(KeyboardSensor, {
            coordinateGetter: sortableKeyboardCoordinates,
        })
    );

    // Custom collision detection
    const customCollisionDetection: CollisionDetection = (args) => {
        const { active } = args;
```

```
        const isDraggingScript = active?.data.current?.type === "script";

        // When dragging scripts, prioritize folder drop zones
        if (isDraggingScript) {
            const pointerCollisions = pointerWithin(args);
            if (pointerCollisions.length > 0) {
                const droppableCollision = pointerCollisions.find(
                    ({ id }) => String(id).startsWith("folder-droppable-")
                );
                if (droppableCollision) {
                    return [droppableCollision];
                }
            }
        }

        // For folders, use normal rect intersection
        return rectIntersection(args);
    };

    const handleDragStart = (event: DragStartEvent) => {
        const { active } = event;
        setActiveId(active.id as number);
        setActiveType(active.data.current?.type || null);

        // Only set reordering state when dragging folders
        if (active.data.current?.type === "folder") {
            dispatch(folderSlice.actions.setIsReorderingFolder(true));
        }
    };

    const handleDragEnd = (event: DragEndEvent) => {
        const { active, over } = event;

        if (!over || !folderResponse || !selectedFolderId) {
            dispatch(folderSlice.actions.setIsReorderingFolder(false));
            return;
        }

        const activeData = active.data.current;
        const overData = over.data.current;

        // Case 1: Script dropped on folder
        if (activeData?.type === "script" && overData?.type === "folder")
{
            moveScript({
                ...activeData.script,
                folderId: overData.folderId,
            });
        }
        // Case 2: Reordering scripts
        else if (activeData?.type === "script" && overData?.type ===
"script") {
            // Reorder logic...
        }
```

```
        // Case 3: Reordering folders
        else if (activeData?.type === "folder" && overData?.type ===
"folder") {
            // Reorder logic...
        }

        dispatch(folderSlice.actions.setIsReorderingFolder(false));
        setActiveId(null);
        setActiveType(null);
    };

    return (
        <DndContext
            sensors={sensors}
            collisionDetection={customCollisionDetection}
            onDragStart={handleDragStart}
            onDragEnd={handleDragEnd}
        >
            <SortableSubfolders folderResponse={folderResponse} />
            <SortableScripts
                folderResponse={folderResponse}
                selectedFolderId={selectedFolderId}
            />

            {/* DragOverlay for smooth animations */}
            <DragOverlay>
                {activeId && activeType === "script" && (
                    <div className="opacity-80">
                        <ScriptItem script={script} folderId={folderId} />
                    </div>
                )}
                {activeId && activeType === "folder" && (
                    <div className="opacity-80">
                        <CollapsableFolder folder={folder} />
                    </div>
                )}
            </DragOverlay>
        </DndContext>
    );
}
```

---

2. Sortable Folder (Dual Purpose: Sortable + Droppable)

File: `src/app-component/ScriptsColumn/CollapsableFolder.tsx`

```
export default function CollapsableFolder({ folder }) {
    // Make folder sortable (for reordering folders)
    const {
        attributes,
        listeners,
```

```
        setNodeRef: setSortableNodeRef,
        transform,
        transition,
        isDragging,
        setActivatorNodeRef,
    } = useSortable({
        id: folder.id,
        data: {
            type: "folder",
            folderId: folder.id,
            folder: folder,
        },
    });

    // Make folder droppable (to accept scripts)
    const { setNodeRef: setDroppableNodeRef, isOver } = useDroppable({
        id: `folder-droppable-${folder.id}`,
        data: {
            type: "folder",
            folderId: folder.id,
            folder: folder,
        },
    });

    // Get active drag context
    const { active } = useDndContext();
    const isDraggingScript = active?.data.current?.type === "script";

    // Only highlight when a script hovers over this folder
    const showHighlight = isOver && isDraggingScript;

    // Combine both refs
    const setNodeRef = (node: HTMLElement | null) => {
        setSortableNodeRef(node);
        setDroppableNodeRef(node);
    };

    const style: React.CSSProperties = {
        transform: CSS.Transform.toString(transform),
        transition: isDragging ? "none" : transition,
        opacity: isDragging ? 0 : 1,  // Hide original when dragging
    };

    return (
        <div ref={setNodeRef} style={style} {...attributes}>
            <div className={clsx({
                "bg-gray-400 dark:bg-neutral-600": showHighlight,
                // Other styles...
            })}>
                <div ref={setActivatorNodeRef} {...listeners}>
                    <GripVertical />
                </div>
                <Folder className="w-4 h-4" fill="currentColor" />
                {folder.name}
```

```
            </div>
        </div>
    );
}
```

Key Points:

- Uses **both** `useSortable` and `useDroppable`
- Combines refs so both hooks work on the same element
- Shows highlight only when dragging a script over it
- Sets `opacity: 0` when being dragged (only overlay visible)

---

3. Sortable Script

File: `src/app-component/ScriptsColumn/SortableScriptItem.tsx`

```tsx
export default function SortableScriptItem({ script, folderId }) {
    const {
        attributes,
        listeners,
        setNodeRef,
        transform,
        transition,
        isDragging,
        setActivatorNodeRef,
    } = useSortable({
        id: script.id,
        data: {
            type: "script",
            script: script,
        },
    });

    const style: React.CSSProperties = {
        transform: CSS.Transform.toString(transform),
        transition: isDragging ? "none" : transition,
        opacity: isDragging ? 0 : 1,  // Hide when dragging
    };

    return (
        <div ref={setNodeRef} style={style} {...attributes}>
            <div ref={setActivatorNodeRef} {...listeners}>
                <GripVertical className="w-4 h-4" />
            </div>
            <ScriptItem script={script} folderId={folderId} />
        </div>
    );
}
```

Key Challenges & Solutions

Challenge 1: Crossing SortableContext Boundaries

Problem: Dragging a script from the scripts list to a folder in the folders list caused animation interruption.

Solution:

1. Use a single `DndContext` wrapping both lists
2. Make folders **both sortable AND droppable**
3. Add `DragOverlay` to show a smooth clone following the cursor

```
// Single DndContext for both lists
<DndContext>
    <SortableContext items={folders}>
        {/* Folders */}
    </SortableContext>
    <SortableContext items={scripts}>
        {/* Scripts */}
    </SortableContext>
    <DragOverlay>
        {/* Smooth clone */}
    </DragOverlay>
</DndContext>
```

Challenge 2: Folders Not Highlighting When Scripts Hover

Problem: The `isReorderingFolder` state was set to `true` for all drag operations, making folders transparent.

Solution: Only set reordering state when dragging **folders**, not scripts:

```
const handleDragStart = (event: DragStartEvent) => {
    const { active } = event;

    // Only set reordering for folders
    if (active.data.current?.type === "folder") {
        dispatch(setIsReorderingFolder(true));
    }
};
```

Challenge 3: Collision Detection Interfering with Folder Sorting

Problem: Custom collision detection was always prioritizing droppable zones, making folder-to-folder sorting difficult.

Solution: Use type-aware collision detection:

```
const customCollisionDetection: CollisionDetection = (args) => {
    const { active } = args;
    const isDraggingScript = active?.data.current?.type === "script";

    // Only prioritize droppables when dragging scripts
    if (isDraggingScript) {
        return pointerWithin(args).filter(/* folders only */);
    }

    // Use normal collision for folders
    return rectIntersection(args);
};
```

Challenge 4: Identifying Drop Targets

Problem: How to distinguish between dropping on a folder vs reordering?

Solution: Use the `data` property to attach metadata:

```
// In useSortable/useDroppable
data: {
    type: "script" | "folder",
    script: scriptObject,
    folderId: number,
}

// In handleDragEnd
const activeData = active.data.current;
const overData = over.data.current;

if (activeData?.type === "script" && overData?.type === "folder") {
    // Script dropped on folder!
}
```

Component Architecture

Data Flow

```
1. User drags script
   ↓
2. handleDragStart
   - Store activeId & activeType
   - Show DragOverlay clone
   ↓
3. Collision detection runs continuously
   - Detects if hovering over folder
```

```
   - Folder highlights (isOver = true)
   ↓
4. User drops script
   ↓
5. handleDragEnd
   - Check types (script + folder?)
   - Call API to move script
   - Hide overlay
   ↓
6. Cache updates optimistically
   - Remove from source folder
   - Add to target folder
```

Best Practices

1. Use Type Metadata

Always attach type information to draggable items:

```
data: {
    type: "script" | "folder",
    // ... other data
}
```

2. Combine Refs for Dual-Purpose Elements

When an element is both sortable and droppable:

```
const setNodeRef = (node) => {
    setSortableNodeRef(node);
    setDroppableNodeRef(node);
};
```

3. Hide Original During Drag

Set opacity to 0 so only the DragOverlay is visible:

```
opacity: isDragging ? 0 : 1;
```

4. Context-Aware Highlighting

Only show highlights for valid drop operations:

```
const { active } = useDndContext();
const isDraggingScript = active?.data.current?.type === "script";
const showHighlight = isOver && isDraggingScript;
```

5. Custom Collision Detection

Implement type-aware collision detection to avoid conflicts:

```
const customCollisionDetection = (args) => {
    const isDraggingScript = args.active?.data.current?.type === "script";

    if (isDraggingScript) {
        return pointerWithin(args); // For dropping
    }

    return rectIntersection(args); // For sorting
};
```

Visual Feedback

Drag States

State	Visual Effect
Normal	`opacity: 1`
Dragging (original)	`opacity: 0` (hidden)
Dragging (overlay)	`opacity: 0.8` (semi-transparent)
Drop target hovered	`bg-gray-400 dark:bg-neutral-600`

Summary

This project uses **@dnd-kit** to implement a sophisticated drag-and-drop system where:

1. ✅ Scripts can be reordered within their list
2. ✅ Scripts can be dragged into folders
3. ✅ Folders can be reordered
4. ✅ Visual feedback shows valid drop targets
5. ✅ Smooth animations throughout

The key to success was:

- Using a single unified `DndContext`
- Making folders both sortable and droppable
- Type-aware collision detection

- Custom metadata in the `data` property
- DragOverlay for smooth visual feedback

Resources

- dnd-kit Documentation
- dnd-kit Examples
- GitHub Repository