React and Vanilla JS

The Fight for DOMination

Samdish Suri
@awreccan
Google Keep

Everything in one place
Any way you access Keep, all your notes stay in sync.

On the web
https://keep.google.com

On Android
https://g.co/keep

In Chrome
https://g.co/keepinchrome

Make a list

- To rearrange your list, drag and drop items
- Check things off when you've finished
- Ticked items automatically move to the bottom of the list and...

And duplicate it!

- To rearrange your list, drag and drop items
- Check things off when you've finished
- Ticked items automatically move to the bottom of the list and...

Welcome to Google Keep
Capture what's on your mind.
Add notes, lists, photos and audio to Keep.

Finished with a note? Use the archive button or swipe it away on Android.
Give it a try! You can always search for it later.
WHAT IF

YOU COULD DRAG ITEMS TO OTHER LISTS?
challenge accepted
WHAT IF I TOLD YOU

THERE'S A LIBRARY FOR THAT
it’s the perfect DOM manipulation plugin

but it’s pure Vanilla JS

without a React wrapper

that shouldn’t be a problem 🤔
Integrating with DOM Manipulation Plugins

React is unaware of changes made to the DOM outside of React. It determines updates based on its own internal representation, and if the same DOM nodes are manipulated by another library, React gets confused and has no way to recover.

This does not mean it is impossible or even necessarily difficult to combine React with other ways of affecting the DOM, you just have to be mindful of what each is doing.
export const lists = [
    {
        id: 'l2',
        name: 'List 2',
        items: [
            { id: 'i3', name: 'Item 3', dragging: false },
            { id: 'i4', name: 'Item 4', dragging: false }
        ]
    },
    {
        id: 'l4',
        name: 'List 4',
        items: [
            { id: 'i7', name: 'Item 7', dragging: false },
            { id: 'i8', name: 'Item 8', dragging: false }
        ]
    },
    {
        id: 'l3',
        name: 'List 3',
        items: [
            { id: 'i5', name: 'Item 5', dragging: false },
            { id: 'i6', name: 'Item 6', dragging: false }
        ]
    },
    {
        id: 'l1',
        name: 'List 1',
        items: [
            { id: 'i1', name: 'Item 1', dragging: false },
            { id: 'i2', name: 'Item 2', dragging: false }
        ]
    }
]
Sunshine and daisies
and unicorns
### Redux Action

- **DRAG_ITEM**
  - Dragging:
    - Unmount the original
    - Copy it
    - Drag it
    - Drop it

- **DROP_ITEM**
  - Dragging:
    - Mount a new node
Changes in the State refresh the UI

User uses the UI to update the State
State (Redux)

Changes in the State refresh the UI

UI (React)

User uses the UI to update the State

User uses Muuri to update the UI

Muuri
Changes in the State refresh the UI

Renders a duplicate item (undraggable)

Sync Muuri’s UI changes to State

User uses the UI to update the State

User uses Muuri to update the UI
<table>
<thead>
<tr>
<th>Redux Action</th>
<th>Redux State</th>
<th>React vDOM</th>
<th>Muuri DOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mount</td>
<td></td>
<td>Render DOM node</td>
<td>Make it draggable</td>
</tr>
<tr>
<td>DRAG_ITEM</td>
<td></td>
<td>Unmount the original</td>
<td>Copy it</td>
</tr>
<tr>
<td></td>
<td>dragging:</td>
<td></td>
<td>Drag it</td>
</tr>
<tr>
<td>DROP_ITEM</td>
<td></td>
<td>Mount a new node</td>
<td>Drop it</td>
</tr>
<tr>
<td>the fix</td>
<td></td>
<td></td>
<td>Make the React node draggable</td>
</tr>
<tr>
<td>Redux Action</td>
<td>Redux State</td>
<td>React vDOM</td>
<td>Muuri DOM</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Mount</td>
<td></td>
<td>Render DOM node</td>
<td>Make it draggable</td>
</tr>
<tr>
<td>DRAG_ITEM</td>
<td>dragging:</td>
<td>Unmount the original</td>
<td>Copy it</td>
</tr>
<tr>
<td></td>
<td>📌 ⬅️ ✅</td>
<td></td>
<td>Drag it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Drop it</td>
</tr>
<tr>
<td>DROP_ITEM</td>
<td>dragging:</td>
<td>Mount a new node</td>
<td>Unmount the dragged node</td>
</tr>
<tr>
<td></td>
<td>📌 ⬅️ ✅</td>
<td></td>
<td>Make the React node draggable</td>
</tr>
<tr>
<td>the fix</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notice how we wrapped `<select>` in an extra `<div>`. This is necessary because Chosen will append another DOM element right after the `<select>` node we passed to it. However, as far as React is concerned, `<div>` always only has a single child. This is how we ensure that React updates won’t conflict with the extra DOM node appended by Chosen. It is important that if you modify the DOM outside of React flow, you must ensure React doesn’t have a reason to touch those DOM nodes.

OR ensure that you bring React (vDOM) and the DOM back in sync
Changes in the State refresh the UI

User uses the UI to update the State

Renders a duplicate item (undraggable)

Sync Muuri’s UI changes to State

Fix this creatively

User uses Muuri to update the UI
The **z-index** values of child stacking contexts **only have meaning in the parent**. Stacking contexts are treated atomically as a single unit in the parent stacking context.
has open menu
Diagnosis

React vDOM

Muuri

React

list list-id

+ MUURI-DRAGGABLE-ITEM

+ muuri-itemShown

+ stack-higher

<MuuriGridItem className="list list-${list.id}"/>

<MuuriGridItem className="list list-${list.id}${list.stackingClass ? ' stack-higher' : ''}"/>
Diagnosis

React vDOM

Muuri

React

+ list list-id

+ MUURI-DRAGGABLE-ITEM

+ muuri-item-shown

+ stack-higher

write

without read

time
Class Warfare

Solutions?
Class Warfare

Solutions?
React and Vanilla JS

The Fight for DOMination

github.com/awreccan/assigner

Samdish Suri
@awreccan
Register early and get the best discounts!