Josh Ellis
Posted on 3 de set. de 2020

CodeAlong: Multi-Column Drag and Drop in React

#tutorial #react #codenewbie #javascript

JustDoThree (2 Part Series)

1 Why I'm Building A Productivity App

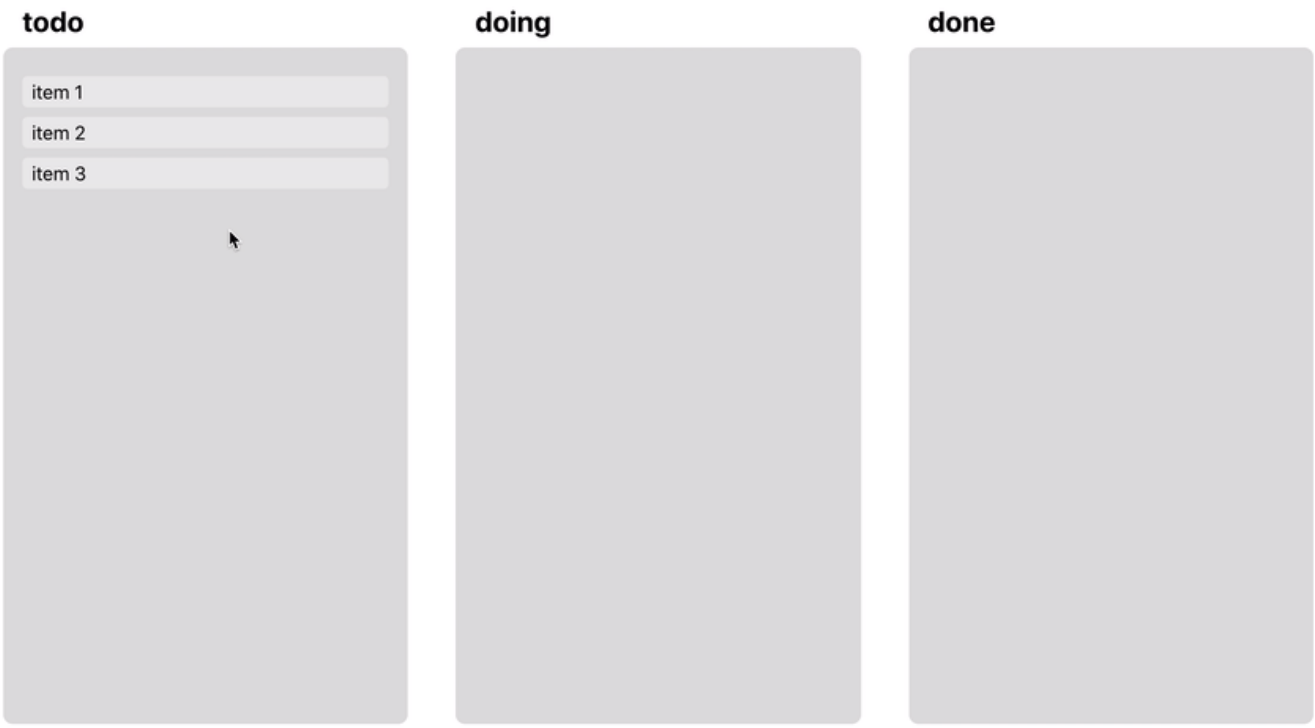
2 **CodeAlong: Multi-Column Drag and Drop in React**

I've been working on a productivity app, and like many of its siblings, drag-and-drop is a key feature for moving items around within the app.

After looking into and trying a few options, I settled on react-beautiful-dnd. Here's a generic example of how to use it to add drag and drop into your own project.

Follow along. Starter code is available on GitHub or Code Sandbox! Just check out the `start here` branch.

✨ What We're Building



Live Demo

Code Sandbox

💾 Installation

To start, create a fresh install of `create-react-app`:

```
npx create-react-app drag-and-drop --typescript
```

The `--typescript` flag is optional, but I prefer TypeScript, so I'll be using that. This should work just fine without TypeScript. We're ready to install react-beautiful-dnd:

```
yarn add react-beautiful-dnd
```

If using TypeScript, you'll want the types as well: `yarn add @types/react-beautiful-dnd`

After the tutorial, I'll be using stitches to make the live demo pretty. This tutorial won't cover styling, but if you want to install stitches, check out the installation docs.

🚀 Getting Started

The first thing to do is clean out `App.tsx`:

```
// src/App.tsx
```

```
// src/App.tsx
import React from 'react'


function App () {
  return <div></div>
}

export default App
```

From here, let's create a simple single-column list using `div`s and temporary, ugly inline styling:

```
// src/App.tsx

function App () {
  return (
    <div
      style={{
        display: 'flex',
        margin: '24px auto',
        maxWidth: '128px',
        flexDirection: 'column',
        justifyContent: 'space-between'
      }}
    >
      <div>Item 1</div>
      <div>Item 2</div>
      <div>Item 3</div>
    </div>
  )
}
```

The result is a very underwhelming and ugly list. Let's keep moving.

⚙️ Components Time

I know it doesn't look like much yet, but things are about to get complicated, so we need to set up components. First, lets make an `<Item>` component and replace the hard-coded list with an array and map:

```
// src/App.tsx
import Item from './components/Item'

function App () {
  const list = ['Item 1', 'Item 2', 'Item 3']
  return (
```

```
    <div
      //...style...
     >
      {list.map((text, index) => (
        <Item key={text} text={text} index={index} />
      ))}
    </div>
  )
}


// src/Item.tsx
import React from 'react'

// TypeScript only
interface ItemProps {
  text: string
  index: number
}

// ": React.FC<ItemProps>" is TypeScript only
const Item: React.FC<ItemProps> = ({ text, index }) => {
  return <div>{text}</div>
}

export default Item
```

Note: We don't need the index yet, but we will.

Next, to prep for multi-column we'll create a `<Column>` component as well:

```
// src/components/Column.tsx
import React from 'react'
import Item from './Item'

// TypeScript only
interface ColumnProps {
  list: string[]
}

// ": React.FC<ItemProps>" is TypeScript only
const Column: React.FC<ColumnProps> = ({ list }) => {
  return (
    <div
      style={{
        display: 'flex',
        flexDirection: 'column'
      }}
```

```
      }}
    >
      {list.map((text, index) => (
        <Item key={text} text={text} index={index} />
      ))}
    </div>
  )
}

export default Column
```

Let's update `App.tsx` with the new component. We'll still only have one column for now, but we can set up the inline styling for a three-column grid while we're at it:

```
// src/App.tsx
import Column from './components/Column'

function App () {
  const list = ['Item 1', 'Item 2', 'Item 3']
  return (
    <div
      style={{
        display: 'grid',
        gridTemplateColumns: '1fr 1fr 1fr',
        margin: '24px auto',
        width: '80%',
        gap: '8px'
      }}
    >
      <Column list={list} />
    </div>
  )
}
```

🔮 Adding Drag and Drop

The moment you've been waiting for.

The `react-beautiful-dnd` package expects a syntax you might not have seen before. It uses components that expect a function as its child. That function then returns JSX/TSX containing the element you want to make droppable/draggable.

This is because the function has an argument called `provided` that needs to be passed as props to the droppable/draggable elements. (It also allows use of a second, more advanced argument that we won't need today.)

This will all make more sense in a min, I promise.

This will all make more sense in a minute...

First, we need to wrap everything in a `<DragDropContext>`. We also need to make a `onDragEnd` function that we'll be using later.

```
// src/App.tsx
import {DragDropContext} from 'react-beautiful-dnd'
/* ... */
  const onDragEnd = () => null

  return (
    <DragDropContext onDragEnd={onDragEnd}>
      <div
        style={{
          display: 'grid',
          gridTemplateColumns: '1fr 1fr 1fr',
          margin: '24px auto',
          width: '80%',
          gap: '8px'
        }}
      >
        <Column list={list} />
      </div>
    </DragDropContext>
  )
  /* ... */
```

Next, we need to convert `Column` into a `<Droppable>`, add `provided.props` and `provided.innerRef` to the returned `<div>` and add `provided.placeholder`:

```
// src/components/Column.tsx
import { Droppable } from 'react-beautiful-dnd'

const Column: React.FC<ColumnProps> = ({ list }) => {
  return (
    <Droppable droppableId='col-1'>
      {provided => (
        <div
          style={{
            display: 'flex',
            flexDirection: 'column'
          }}
          {...provided.droppableProps}
          ref={provided.innerRef}
        >
          {list.map((text, index) => (
```

```
            <Item key={text} text={text} index={index} />
          ))}

          {provided.placeholder}
        </div>
      )}
    </Droppable>
  )
}
```

Finally, we do a similar thing with `Item`, turning it into a `<Draggable>` and adding `provided.innerRef`, `provided.dragHandleProps`, and `provided.draggableProps` to the `div`:

```
// src/components/Item.tsx
import { Draggable } from 'react-beautiful-dnd'

const Item: React.FC<ItemProps> = ({ text, index }) => {
  return (
    <Draggable draggableId={text} index={index}>
      {provided => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
        >
          {text}
        </div>
      )}
    </Draggable>
  )
}
```

The `provided.dragHandleProps` allows you to have a specific part of the element be the handle, but we're going to keep it simple and have the whole element as the handle.

🎉 It works! But not really

So by now you'll have something that looks like this:

The items are draggable and they seem to move around correctly, but when you drop an item, everything just goes back to how it was. That's because we haven't introduced `state` into the equation yet. Remember `onDragEnd`?

That function is executed exactly when you'd expect: at the end of the drag. It has access to source and destination objects, which have useful information for updating our state.

But first, let's make our list stateful with a `useState()` hook:

```
// src/App.tsx
/* ... */
const [list, setList] = useState(['Item 1', 'Item 2', 'Item 3'])
/* ... */
```

If you're not familiar with hooks, check out the official docs.

Our goal is to update the list every time an item is dropped, so let's start writing `onDragEnd`:

```
// src/App.tsx
/* ... */
const [list, setList] = useState(['Item 1', 'Item 2', 'Item 3'])

  const onDragEnd = ({ source, destination }: DropResult) => {
    // Make sure we have a valid destination
    if (destination === undefined || destination === null) return null

    // Make sure we're actually moving the item
    if (destination.index === source.index) return null

    // Move the item within the list
    // Start by making a new list without the dragged item
    const newList = list.filter((_: any, idx: number) => idx !== source.index)

    // Then insert the item at the right location
    newList.splice(destination.index, 0, list[source.index])

    // Update the list
    setList(newList)
  }
/* ... */
```

The comments in that snippet are hopefully self explanatory. The list now retains its order!

If all you need is a single list, you're done!

🕳 Multiple Drop Zones

But we're not done here yet! Let's add a few more columns.

First, we need to upgrade the way we keep track of state:

```
// src/App.tsx
/* ... */
const initialColumns = {
    todo: {
      id: 'todo',
      list: ['item 1', 'item 2', 'item 3']
    },
    doing: {
      id: 'doing',
      list: []
    },
    done: {
      id: 'done',
      list: []
    }
  }
  const [columns, setColumns] = useState(initialColumns)
  /*...*/
```

As you can see, we now have three columns, each with an id and its own list. We'll use the IDs in a minute. In a more complex app, you might also have a `title` field on each column and use a different kind of ID, but we're keeping it simple for now.

Let's update `App`'s return to map through the columns:

```
// src/App.tsx
/* ... */
```

```
  {Object.values(columns).map(col => (
    <Column col={col} key={col.id} />

  ))}
/* ... */
```

We changed the props to just be `col` because I prefer to just destructure the object on the other end:

```
// src/components/Column.tsx
/* ... */
// TypeScript only
interface ColumnProps {
  col: {
    id: string
    list: string[]
  }
}

const Column: React.FC<ColumnProps> = ({ col: { list, id } }) => {
  return (
    <Droppable droppableId={id}>
      {provided => (
        <div
          style={{
            display: 'flex',
            flexDirection: 'column',
          }}
          >
          <h2>{id}</h2>
          <div
            style={{
              display: 'flex',
              flexDirection: 'column',
              minHeight: '120px'
            }}
            {...provided.droppableProps}
            ref={provided.innerRef}
          >
            {list.map((text, index) => (
              <Item key={text} text={text} index={index} />
            ))}
            {provided.placeholder}
          </div>
        </div>
      )}
    </Droppable>
  )
```

```
}
/* ... */
```

Note: I added a header here, so I had to adjust the `div`s a bit.

When using multiple columns, it's important to have a minimum height on the element that takes `provided.droppableProps`.

😢 We Broke It (again)

You should now have three columns, and the first column should have three items in it. Dragging works, but the items go back to where they were.

That's because we need to update the `onDragEnd` function to handle our new setup.

First, let's update the single column case:

```
// src/App.tsx
/* ... */
const onDragEnd = ({ source, destination }: DropResult) => {
    // Make sure we have a valid destination
    if (destination === undefined || destination === null) return null

    // If the source and destination columns are the same
    // AND if the index is the same, the item isn't moving
    if (
      source.droppableId === destination.droppableId &&
      destination.index === source.index
    )
      return null


    // Set start and end variables
    const start = columns[source.droppableId]
    const end = columns[destination.droppableId]

    // If start is the same as end, we're in the same column
    if (start === end) {
      // Move the item within the list
      // Start by making a new list without the dragged item
      const newList = start.list.filter(
        (_: any, idx: number) => idx !== source.index
      )

      // Then insert the item at the right location
      newList.splice(destination.index, 0, start.list[source.index])

      // Then create a new copy of the column object
```

```
      // Then create a new copy of the column object
      const newCol = {
        id: start.id,
        list: newList
      }

      // Update the state
      setColumns(state => ({ ...state, [newCol.id]: newCol }))
      return null
    }
    return null
  }
/* ... */
```

Again, the comments should explain the above code. Note: make sure you updated the second `if` block!

If all is well, single column drag/drop should be working now.

Finally, let's set up multi-column drag and drop:

```
// src/App.tsx
/* ... */
const onDragEnd = ({ source, destination }: DropResult) => {

    /* ... */

    // If start is the same as end, we're in the same column
    if (start === end) {
      /* ... */
    } else {
      // If start is different from end, we need to update multiple columns
      // Filter the start list like before
      const newStartList = start.list.filter(
        (_: any, idx: number) => idx !== source.index
      )

      // Create a new start column
      const newStartCol = {
        id: start.id,
        list: newStartList
      }

      // Make a new end list array
      const newEndList = end.list

      // Insert the item into the end list
      newEndList.splice(destination.index, 0, start.list[source.index])
```

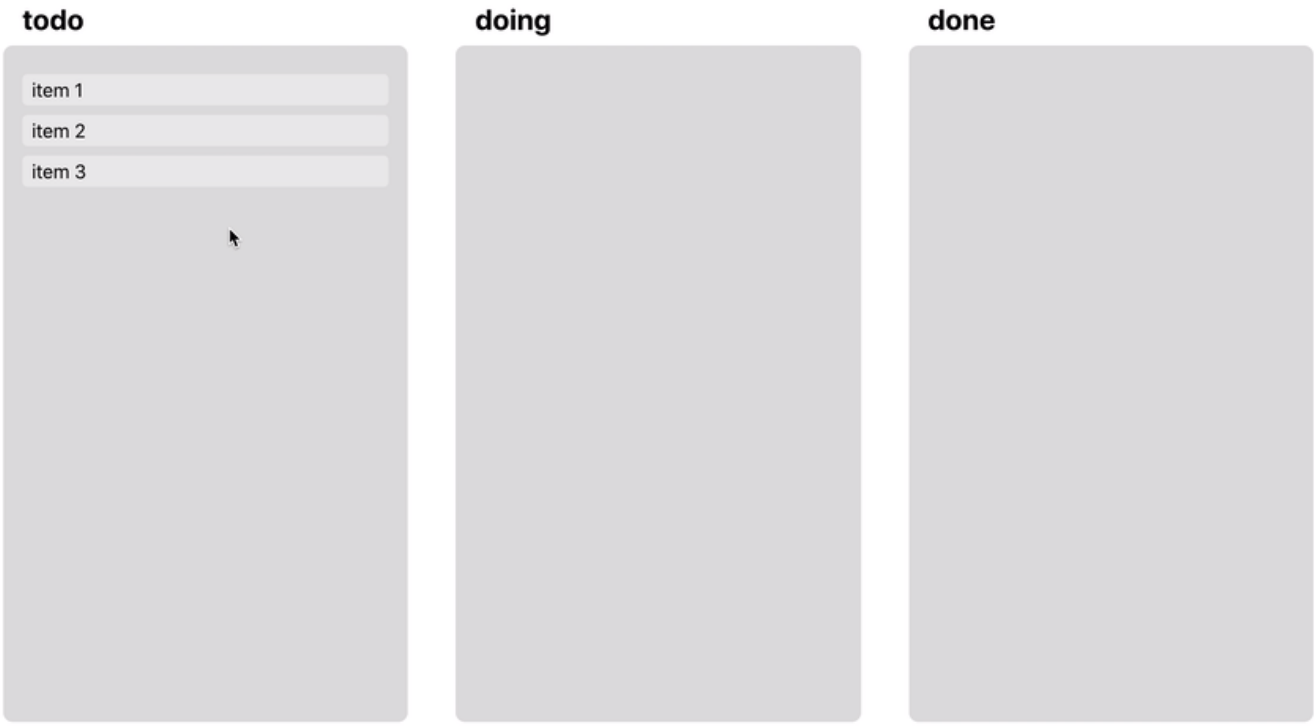
```
      // Create a new end column
      const newEndCol = {
        id: end.id,
        list: newEndList
      }

      // Update the state
      setColumns(state => ({
        ...state,
        [newStartCol.id]: newStartCol,
        [newEndCol.id]: newEndCol
      }))
      return null
    }
  }
/* ... */
```

That's a decent chunk of code, but the idea is simple. We're updating the lists of each column, then updating the state.

🥳 It Works!

The final product works perfectly! With a little styling, you might end up with something like this:



Live Demo • Code Sandbox

Let's Talk

If you made it this far, thanks for reading! Will you be using `react-beautiful-dnd` in the future?

I'd love to see what you make.

Consider following me here on Dev.to if you want more content like this.

Next week, I'll be back with another article related to my upcoming productivity app. Stay tuned!

JustDoThree (2 Part Series)

1 Why I'm Building A Productivity App

2 **CodeAlong: Multi-Column Drag and Drop in React**

Discussion (7)

aliplutus • Oct 4 '20

Do you know how to create recursive multi-columns?
in the following code sandbox I tried to create a sortable tree with `react-beautiful-dnd` but I don't know to finish it.
codeSandBox

Josh Ellis • Oct 4 '20 • Edited on Oct 4

I haven't done recursive, so I'm not 100% sure what you're looking for.

But does this example of vertical nested lists help (I didn't make this, it's from the official docs/repo)?

Demo - Code on GH

aliplutus • Oct 4 '20 • Edited on Oct 4

recursive tree means that you can drag any element and drop it inside other elements.

dev-to-uploads.s3.amazonaws.com/i/...

Gustavo Leal • Dec 12 '20

I tried to use on typescript but I got this error when I call columns.
// Set start and end variables App.tsx
const start = columns[source.droppableId]
const end = columns[destination.droppableId]
TypeScript error in /home/gustavo/drag-and-drop/src/App.tsx(37,19):
Element implicitly has an 'any' type because expression of type 'string' can't be used to index type '{ todo: { id: string; list: string[]; }; doing: { id: string; list: never[]; }; done: { id: string; list: never[]; }; }'.
No index signature with a parameter of type 'string' was found on type '{ todo: { id: string; list: string[]; }; doing: { id: string; list: never[]; }; done: { id: string; list: never[]; }; }'. TS7053.

Some idea how to fix it ?

Josh Ellis • Dec 13 '20

If you show me your repo, I can give you a more specific answer, but I think what's happening is Typescript thinks the object key should be from a very specific list ("todo"|"doing"|"done", for example) but you (or the library) are giving it a string...

See this for some examples of how to fix: stackoverflow.com/questions/568334...

Gustavo Leal • Dec 14 '20

Actually I tried to add your DND on my profile project, but I got this error. So I copy and pasted your code for try to understand if it was an under spelling mistake but I got the same error.
github.com/gustavojleal/dnd-react-....

Thanks so much for your time,

Uros Randelovic • Mar 9 '21

Really good walkthrough mate!

Josh Ellis

he/him • Crafting products that simplify the human experience. I'm passionate about innovative tech in productivity 📋, mental health 🧘, and travel 🏕.

LOCATION
Chicago

EDUCATION
Software Engineering @ Flatiron School

WORK
Frontend Developer at Endeavor Business Media

JOINED
25 de fev. de 2020

More from Josh Ellis

How to Setup MongoDB with Go
#mongodb #go #tutorial

Quick Tip: How to Enable Syntax Highlighting on Dev.to
#beginners #writing #codenewbie

Simple Download Text File Component with React
#javascript #typescript #react #webdev