Sorting

We've covered most of React's fundamentals, its legacy features, and techniques for maintaining applications. Now it's time to dive into developing real-world React applications. Each of the following sections will come with a task. Try to tackle these tasks without the *optional hints* first, but be aware that these are going to be challenging on your first attempt. If you need help, use the *optional hints* or follow the instructions from the section.

Task: Working with a list of items often includes interactions that make data more approachable by users. So far, every item was listed with each of its properties. To make it explorable, the list should enable sorting of each property by title, author, comments, and points in ascending or descending order. Sorting in only one direction is fine, because sorting in the other direction will be part of the next section.

Optional Hints:

- Introduce a new sort state in the App or List component.
- For each property (e.g. `title`, `author`, `points`, `num_comments`) implement an HTML button which sets the sort state for this property.
- Use the sort state to apply an appropriate sort function on the `list`.
- Using a utility library like Lodash for its `sortBy` function is encouraged.

We will treat the list of data like a table. Each row represents an item of the list and each column represents one property of the item. Headers provide the user more guidance about each column:

```
const List = ({ list, onRemoveItem }) => (
  <div>
    <div style={{ display: 'flex' }}>
      <span style={{ width: '40%' }}>Title</span>
      <span style={{ width: '30%' }}>Author</span>
      <span style={{ width: '10%' }}>Comments</span>
```

```
      <span style={{ width: '10%' }}>Points</span>
      <span style={{ width: '10%' }}>Actions</span>
    </div>

    {list.map(item => (
      <Item
        key={item.objectID}
        item={item}
        onRemoveItem={onRemoveItem}
      />

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

src/App.js

We are using inline style for the most basic layout. To match the layout of the header with the rows, give the rows in the Item component a layout as well:

```
const Item = ({ item, onRemoveItem }) => (

  <div style={{ display: 'flex' }}>
    <span style={{ width: '40%' }}>
      <a href={item.url}>{item.title}</a>
    </span>

    <span style={{ width: '30%' }}>{item.author}</span>
    <span style={{ width: '10%' }}>{item.num_comments}</span>
    <span style={{ width: '10%' }}>{item.points}</span>
    <span style={{ width: '10%' }}>

      <button type="button" onClick={() => onRemoveItem(item)}>
        Dismiss
      </button>
    </span>
  </div>
);
```

src/App.js

In the ongoing implementation, we will remove the style attributes, because it takes up lots of space and clutters the actual implementation logic (hence extracting it into proper CSS). But I encourage you to keep it for yourself.

The List component will handle the new sort state. This can also be done in the App component, but only the List component is in play, so we can lift the state management directly to it. The sort state initializes with a `'NONE'` state, so the list items are displayed in the order they are fetched from the API. Further, we added a new handler to set the sort state with a sort-specific key.

```
const List = ({ list, onRemoveItem }) => {
  const [sort, setSort] = React.useState('NONE');


  const handleSort = sortKey => {
    setSort(sortKey);
  };

  return (

    ...
  );
};
```

src/App.js

In the List component's header, buttons can help us to set the sort state for each column/property. An inline handler is used to sneak in the sort-specific key (`sortKey`). When the button for the "Title" column is clicked, `'TITLE'` becomes the new sort state.

```
const List = ({ list, onRemoveItem }) => {
  ...

  return (
    <div>
      <div>
        <span>
          <button type="button" onClick={() => handleSort('TITLE')}>
            Title
          </button>

        </span>
        <span>
          <button type="button" onClick={() => handleSort('AUTHOR')}>
            Author
          </button>

        </span>
        <span>

          <button type="button" onClick={() => handleSort('COMMENT')}>
            Comments
          </button>

        </span>
        <span>
          <button type="button" onClick={() => handleSort('POINT')}>
            Points
          </button>
        </span>
        <span>Actions</span>
      </div>

      {list.map(item => ... )}
    </div>
  );
```

```
};
```

src/App.js

State management for the new feature is implemented, but we don't see anything when our buttons are clicked yet. This happens because the sorting mechanism hasn't been applied to the actual `list`.

Sorting an array with JavaScript isn't trivial, because every JavaScript primitive (e.g. string, boolean, number) comes with edge cases when an array is sorted by its properties. We will use a library called Lodash to solve this, which comes with many JavaScript utility functions (e.g. `sortBy`). First, install it via the command line:

```
npm install lodash
```

Second, at the top of your file, import the utility function for sorting:

```
import React from 'react';
import axios from 'axios';
import { sortBy } from 'lodash';

...
```

src/App.js

Third, create a JavaScript object (also called dictionary) with all the possible `sortKey` and sort function mappings. Each specific sort key is mapped to a function that sorts the incoming `list`. Sorting by `'NONE'` returns the unsorted list; sorting by `'POINT'` returns a list and its items sorted by the `points` property.

```
const SORTS = {
  NONE: list => list,
  TITLE: list => sortBy(list, 'title'),
  AUTHOR: list => sortBy(list, 'author'),
  COMMENT: list => sortBy(list, 'num_comments').reverse(),
  POINT: list => sortBy(list, 'points').reverse(),
};


const List = ({ list, onRemoveItem }) => {
  ...
};
```

src/App.js

With the `sort` (`sortKey`) state and all possible sort variations with `SORTS` at our disposal, we can sort the list before mapping it over each Item component:

```
const List = ({ list, onRemoveItem }) => {
  const [sort, setSort] = React.useState('NONE');

  const handleSort = sortKey => {
    setSort(sortKey);
  };

  const sortFunction = SORTS[sort];
  const sortedList = sortFunction(list);

  return (
    <div>
      ...

      {sortedList.map(item => (
        <Item
          key={item.objectID}
          item={item}
          onRemoveItem={onRemoveItem}
        />
      ))}
    </div>
  );
};
```

src/App.js

It's done. First we extracted the sort function from the dictionary by its `sortKey` (state). Then, we applied the function to the list, before mapping it to render each Item component. Again, the initial sort state will be `'NONE'`, meaning it will sort nothing.

Second we rendered more HTML buttons to give our users interaction. Then, we added implementation details for each button by changing the sort state. Finally, we used the sort state to sort the actual list.

```
  □ □ □□  □   ã□  F   □□  □   □  )□     □   9□  5□  @@  □   °□  n□  □PNG
□
   IHDR   □   □□□   (-□S   äPLTE"""""""""""""""""""2PX=r□)7;*:>H□¤-BGE□□8do5Xb6[eK□®K□¯1MU
□
   IHDR   □   □□□   ×©ÍÊ  □ePLTE""""""""""""""""""""""""2RZN¢¹J□«3R[J□¬)59YÁÞ0KS4W`Q«ÄL□²%
?^q÷ñíÛ□ï.},□ìsæÝ_TttÔ¾ □1#□□/(ì□-[□□□è`□è`Ì□ÚïÅðZ□d5□□□□?ÎebZ¿Þ□i.Ûæ□□□ìqÎ□+1°□}Â□5
□
   IHDR       □□   D¤□Æ  □APLTE   """""""""""""""""""""""""""""2RZVºÖ_ÔôU·Ñ=r□$()'25]ÎíC□□0
□
   IHDR   @   @□□   □·□ì  □:PLTE   """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

```
¢ßqÇ8Ù□´□mKË±mÆ¶mÛü·yi!è□ÎªYÏuë ÀÏ_Àï?i÷□ý+ò□□ÄA□|□ù{□□´?¿□_En□).□JËD¤<□
©¬¢Z\Ts©R*□(□  ¯©□J□□□□u□X/□4J□9□¡5·DEµ4kÇ4□&i¥V4Ú□¡®Ð□□¯□vsf:àg,□¢èBC»î$¶□ºÍùî□□á□@□
-ê>Û□º«¢XÕ¢î}ß¨ëÛÑ;□ÃöN´□ØvÅý□Î¸ÿ1 □ë×ÄO@&v/Äþ_□ö\ô□Ç\í.□□¼+0□□;□□□!□fÊ□¦´Ó%Â JY·O□Â□'
```

Exercises:

- Confirm the changes from the last section.
- Read more about Lodash.
- Why did we use numeric properties like `points` and `num_comments` a reverse sort?
- Use your styling skills to give the user feedback about the current active sort. This mechanism can be as straightforward as giving the active sort button a different color.