Advanced Sorting

In this lesson, you'll learn an advanced data sorting technique to sort our app's table by column

Sorting our table component by column

We implemented a client and server-side search interaction earlier. Since you have a Table component, it makes sense to enhance it with advanced interactions. Next, we'll introduce a sort functionality for each column by using the column headers of the Table.

It is possible to write your own sort function, but I prefer to use a utility library like Lodash for these cases. There are other options, but this is the one I prefer. We have Lodash running on the code widgets in this course but if you are following along on a local setup, install Lodash for our sort function:

```
npm install lodash
```

Now we import the sort functionality of Lodash to our *App.js* file:

```
import React, { Component } from 'react';
import fetch from 'isomorphic-fetch';
import { sortBy } from 'lodash';
import './App.css';
```

Creating `SORTS` to reference any function

Now we have several columns in Table: title, author, comments and points columns. You can define sort functions where each takes a list and returns a list of items sorted by a specific property. Additionally, you will need a default sort function that doesn't sort, but returns the unsorted list. This will be the initial state.

```
...

const SORTS = {
```

```
  NONE: list => list,
  TITLE: list => sortBy(list, 'title'),
  AUTHOR: list => sortBy(list, 'author'),

  COMMENTS: list => sortBy(list, 'num_comments').reverse(),
  POINTS: list => sortBy(list, 'points').reverse(),
};

class App extends Component {
  ...
}

...
```

Two of the sort functions return a reversed list. That's to see the items with the highest comments and points, rather than the items with the lowest counts when the list is sorted for the first time.

The `SORTS` object allows you to reference any sort function now.

Again, the App component is responsible for storing the state of the sort. The initial state will be the default sort function, which doesn't sort at all and returns the input list as output.

```
this.state = {
  results: null,
  searchKey: '',
  searchTerm: DEFAULT_QUERY,
  error: null,
  isLoading: false,
  sortKey: 'NONE',
};
```

Once we choose a different `sortKey`, like the `AUTHOR` key, we sort the list with the appropriate sort function from the `SORTS` object.

Setting `sortKey` in local state

Now we define a new class method in App component that sets a `sortKey` to the local component state, then `sortKey` can be used to retrieve the sorting function to apply it to the list:

```
class App extends Component {
  _isMounted = false;

  constructor(props) {

    ...

    this.needsToSearchTopStories = this.needsToSearchTopStories.bind(this);
```

```
    this.setSearchTopStories = this.setSearchTopStories.bind(this);
    this.fetchSearchTopStories = this.fetchSearchTopStories.bind(this);
    this.onSearchSubmit = this.onSearchSubmit.bind(this);
    this.onSearchChange = this.onSearchChange.bind(this);
    this.onDismiss = this.onDismiss.bind(this);
    this.onSort = this.onSort.bind(this);
  }

  ...

  onSort(sortKey) {
    this.setState({ sortKey });
  }

  ...

}
```

Passing `sortKey` to the table component

The next step is to pass the method and `sortKey` to the Table component.

```
class App extends Component {

  ...

  render() {
    const {
      searchTerm,
      results,
      searchKey,
      error,
      isLoading,
      sortKey
    } = this.state;

    ...

    return (
      <div className="page">
        ...
        <Table
          list={list}
          sortKey={sortKey}
          onSort={this.onSort}
          onDismiss={this.onDismiss}
        />
        ...
      </div>
    );
  }
}
```

The Table component is responsible for sorting the list. It takes one of the `SORT` functions by `sortKey` and passes the list as input, after which it keeps mapping over the sorted list.

mapping over the sorted list.

```
const Table = ({
  list,
  sortKey,
  onSort,
  onDismiss
}) =>
  <div className="table">
    {SORTS[sortKey](list).map(item =>
      <div key={item.objectID} className="table-row">
        ...
      </div>
    )}
  </div>
```

In theory, the list should get sorted by one of the functions. But the default sort is set to `NONE`, so nothing is sorted yet, as nothing executes the `onSort()` method to change the `sortKey`.

Implementing sort in the Table

We extend the Table with a row of column headers that use Sort components in columns to sort each column:

```
const Table = ({
  list,
  sortKey,
  onSort,
  onDismiss
}) =>
  <div className="table">
    <div className="table-header">
      <span style={{ width: '40%' }}>
        <Sort
          sortKey={'TITLE'}
          onSort={onSort}
        >
          Title
        </Sort>
      </span>
      <span style={{ width: '30%' }}>
        <Sort
          sortKey={'AUTHOR'}
          onSort={onSort}
        >
          Author
        </Sort>
      </span>
      <span style={{ width: '10%' }}>
        <Sort
          sortKey={'COMMENTS'}
          onSort={onSort}
        >
```

```
          Comments
        </Sort>
      </span>

      <span style={{ width: '10%' }}>
        <Sort
          sortKey={'POINTS'}
          onSort={onSort}
        >
          Points
        </Sort>
      </span>
      <span style={{ width: '10%' }}>
        Archive
      </span>
    </div>
    {SORTS[sortKey](list).map(item =>
      ...
    )}
  </div>
```

Each Sort component gets a specific `sortKey` and the general `onSort()` function. Internally, it calls the method with the `sortKey` to set the specific key.

```
const Sort = ({ sortKey, onSort, children }) =>
  <Button onClick={() => onSort(sortKey)}>
    {children}
  </Button>
```

As you can see, the Sort component reuses your common Button component. On a button click, each individual passed `sortKey` is set by the `onSort()` method, so the list is sorted when column headers are selected.

Now we'll improve the look of the button in the column header. Let's give it a proper `className`:

```
const Sort = ({ sortKey, onSort, children }) =>
  <Button
    onClick={() => onSort(sortKey)}
    className="button-inline"
  >
    {children}
```

This was done to improve the UI. In the next lesson, we'll continue to build upon our search functionality. We'll first reverse-sort our table.

Our app so far:

```
import React, { Component } from 'react';
import { sortBy } from 'lodash';
import classNames from 'classnames';
require('./App.css');

const DEFAULT_QUERY = 'redux';
const DEFAULT_HPP = '100';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';

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

class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      results: null,
      searchKey: '',
      searchTerm: DEFAULT_QUERY,
      error: null,
      isLoading: false,
      sortKey: 'NONE',
    };

    this.needsToSearchTopstories = this.needsToSearchTopstories.bind(this);
    this.setSearchTopstories = this.setSearchTopstories.bind(this);
    this.fetchSearchTopstories = this.fetchSearchTopstories.bind(this);
    this.onSearchChange = this.onSearchChange.bind(this);
    this.onSearchSubmit = this.onSearchSubmit.bind(this);
    this.onDismiss = this.onDismiss.bind(this);
    this.onSort = this.onSort.bind(this);
  }

  onSort(sortKey) {
    this.setState({ sortKey });
  }

  needsToSearchTopstories(searchTerm) {
    return !this.state.results[searchTerm];
  }

  setSearchTopstories(result) {
    const { hits, page } = result;
    const { searchKey, results } = this.state;

    const oldHits = results && results[searchKey]
      ? results[searchKey].hits
      : [];

    const updatedHits = [
      ...oldHits,
```

```
      ...hits
    ];

    this.setState({
      results: {
        ...results,
        [searchKey]: { hits: updatedHits, page }
      },
      isLoading: false
    });
  }

  fetchSearchTopstories(searchTerm, page = 0) {
    this.setState({ isLoading: true });

    fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}`)
      .then(response => response.json())
      .then(result => this.setSearchTopstories(result))
        .catch(e => this.setState({ error: e }));
  }

  componentDidMount() {
    const { searchTerm } = this.state;
    this.setState({ searchKey: searchTerm });
    this.fetchSearchTopstories(searchTerm);
  }

  onSearchChange(event) {
    this.setState({ searchTerm: event.target.value });
  }

  onSearchSubmit(event) {
    const { searchTerm } = this.state;
    this.setState({ searchKey: searchTerm });

    if (this.needsToSearchTopstories(searchTerm)) {
      this.fetchSearchTopstories(searchTerm);
    }

    event.preventDefault();
  }

  onDismiss(id) {
    const { searchKey, results } = this.state;
    const { hits, page } = results[searchKey];

    const isNotId = item => item.objectID !== id;
    const updatedHits = hits.filter(isNotId);

    this.setState({
      results: {
        ...results,
        [searchKey]: { hits: updatedHits, page }
      }
    });
  }

  render() {
    const {
      searchTerm,
      results,
      searchKey,
```

```
      error,
      isLoading,
      sortKey,
    } = this.state;

    const page = (
      results &&
      results[searchKey] &&
      results[searchKey].page
    ) || 0;

    const list = (
      results &&
      results[searchKey] &&
      results[searchKey].hits
    ) || [];

    return (
      <div className="page">
        <div className="interactions">
          <Search
            value={searchTerm}
            onChange={this.onSearchChange}
            onSubmit={this.onSearchSubmit}
          >
            Search
          </Search>
        </div>
        { error
          ? <div className="interactions">
            <p>Something went wrong.</p>
          </div>
          : <Table
                list={list}
                  sortKey={sortKey}
              onSort={this.onSort}
            onDismiss={this.onDismiss}
                />
        }
        <div className="interactions">
          <ButtonWithLoading
            isLoading={isLoading}
            onClick={() => this.fetchSearchTopstories(searchKey, page + 1)}>
            More
          </ButtonWithLoading>
        </div>
      </div>
    );
  }
}

const Search = ({
  value,
  onChange,
  onSubmit,
  children
}) =>
  <form onSubmit={onSubmit}>
    <input
      type="text"
      value={value}
      onChange={onChange}
```

```
    />
    <button type="submit">
      {children}
    </button>
  </form>

const Table = ({
  list,
  sortKey,
  onSort,
  onDismiss
}) => {
  const sortedList = SORTS[sortKey](list);

  return(
    <div className="table">
      <div className="table-header">
        <span style={{ width: '40%' }}>
          <Sort
            sortKey={'TITLE'}
            onSort={onSort}
          >
            Title
          </Sort>
        </span>
        <span style={{ width: '30%' }}>
          <Sort
            sortKey={'AUTHOR'}
            onSort={onSort}
          >
            Author
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'COMMENTS'}
            onSort={onSort}
          >
            Comments
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          <Sort
            sortKey={'POINTS'}
            onSort={onSort}
          >
            Points
          </Sort>
        </span>
        <span style={{ width: '10%' }}>
          Archive
        </span>
      </div>
      { SORTS[sortKey](list).map(item =>
        <div key={item.objectID} className="table-row">
          <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
              onClick={() => onDismiss(item.objectID)}
              className="button-inline"
            >
              Dismiss
            </Button>
          </span>
        </div>      )}
    </div>
  );
}

const Button = ({ onClick, className = '', children }) =>
  <button
    onClick={onClick}
    className={className}
    type="button"
  >
    {children}
  </button>

const Loading = () =>
  <div>Loading ...</div>

const withLoading = (Component) => ({ isLoading, ...rest }) =>
  isLoading ? <Loading /> : <Component { ...rest } />

const ButtonWithLoading = withLoading(Button);

const Sort = ({
  sortKey,
  activeSortKey,
  onSort,
  children
}) => {
  const sortClass = classNames(
    'button-inline',
  );

  return (
    <Button
      onClick={() => onSort(sortKey)}
      className={sortClass}
    >
      {children}
    </Button>
  );
}

export default App;

export {
  Button,
  Search,
  Table,
};
```