ES6 Spread Operators

We will further extend our application here by modifying the onDismiss() method to tackle with the result object.

The "Dismiss" button doesn't work because the `onDismiss()` method is not aware of the complex result object. It only knows about a plain list in the local state. But it isn't a plain list anymore. Let's change it to operate on the result object instead of the list itself.

```
onDismiss(id) {
  const isNotId = item => item.objectID !== id;
  const updatedHits = this.state.result.hits.filter(isNotId);
  this.setState({
    ...
  });
}
```

In `setState()`, the result is now a complex object, and the list of hits is only one of multiple properties in the object. Only the list gets updated when an item gets removed in the result object, though, while the other properties stay the same.

We could alleviate this challenge by mutating the hits in the result object. It is demonstrated below for the sake of understanding, but we'll end up using another way.

```
// don`t do this
this.state.result.hits = updatedHits;
```

As we know, React embraces immutable data structures, so we don't want to mutate an object (or mutate the state directly). We want to generate a new object based on the information given, so none of the objects get altered and we keep the immutable data structures. You will always return a new object, but never alter the original object.

For this, we use JavaScript ES6's `Object.assign()`. It takes a target object as first argument. All following arguments are source objects, which are merged

into the target object. The target object can be an empty object. It embraces immutability, because no source object gets mutated.

```
const updatedHits = { hits: updatedHits };
const updatedResult = Object.assign({}, this.state.result, updatedHits);
```

Source objects will override previously merged objects with the same property names. Let's do this on the `onDismiss()` method:

```
onDismiss(id) {
  const isNotId = item => item.objectID !== id;
  const updatedHits = this.state.result.hits.filter(isNotId);
  this.setState({
    result: Object.assign({}, this.state.result, { hits: updatedHits })
  });
}
```

That's one solution, but there is a simpler way in JavaScript ES6, called the *spread operator*. The spread operator is shown with three dots: `...` When it is used, every value from an array or object gets copied to another array or object.

We'll examine the ES6 **array** spread operator even though you don't need it yet:

```
const userList = ['Robin', 'Andrew', 'Dan'];
const additionalUser = 'Jordan';
const allUsers = [ ...userList, additionalUser ];

console.log(allUsers);
// output: ['Robin', 'Andrew', 'Dan', 'Jordan']
```

The `allUsers` variable is a completely new array. The other variables `userList` and `additionalUser` stay the same. You can even merge two arrays into a new array.

```
const oldUsers = ['Robin', 'Andrew'];
const newUsers = ['Dan', 'Jordan'];
const allUsers = [ ...oldUsers, ...newUsers ];

console.log(allUsers);
```

Now let's have a look at the object spread operator, which is not JavaScript ES6. It is part of a proposal for a next JavaScript version that is already being used by the React community, which is why *create-react-app* incorporated the feature in the configuration. The object spread operator works just like the JavaScript ES6 array spread operator, except with objects. Each key value pair is copied into a new object:

```
const userNames = { firstname: 'Robin', lastname: 'Wieruch' };
const age = 28;
const user = { ...userNames, age };

console.log(user);
// output: { firstname: 'Robin', lastname: 'Wieruch', age: 28 }
```

Multiple objects can be spread, as with the array spread example.

```
const userNames = { firstname: 'Robin', lastname: 'Wieruch' };
const userAge = { age: 28 };
const user = { ...userNames, ...userAge };

console.log(user);
// output: { firstname: 'Robin', lastname: 'Wieruch', age: 28 }
```

It can be used to replace `Object.assign()`:

```
onDismiss(id) {
  const isNotId = item => item.objectID !== id;
  const updatedHits = this.state.result.hits.filter(isNotId);
  this.setState({
    result: { ...this.state.result, hits: updatedHits }
  });
}
```

Q

In React, you can combine two objects using Spread Operator and add extra properties to that object too. Is this statement True or False?

COMPLETED 0%

1 of 1

The “Dismiss” button should work now, because the `onDismiss()` method is aware of the complex result object, and how to update it after dismissing an item from the list.

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

const DEFAULT_QUERY = 'redux';

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

const isSearched = (searchTerm) => (item) =>
  item.title.toLowerCase().includes(searchTerm.toLowerCase());

class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      result: null,
      searchTerm: DEFAULT_QUERY,
    };

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

  setSearchTopstories(result) {
    this.setState({ result });
  }

  fetchSearchTopstories(searchTerm) {
    fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}`)
      .then(response => response.json())
      .then(result => this.setSearchTopstories(result))
      .catch(e => e);
  }

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

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

  onDismiss(id) {
    const isNotId = item => item.objectID !== id;
    const updatedHits = this.state.result.hits.filter(isNotId);
    this.setState({
      result: { ...this.state.result, hits: updatedHits }
    });
  }

  render() {
    const { searchTerm, result } = this.state;

    if (!result) { return null; }

    return (
      <div className="page">
        <div className="interactions">
          <Search
            value={searchTerm}
            onChange={this.onSearchChange}
          >
            Search
          </Search>
        </div>
        <Table
          list={result.hits}
          pattern={searchTerm}
          onDismiss={this.onDismiss}
        />
      </div>
    );
  }
}


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

const Table = ({ list, pattern, onDismiss }) =>
  <div className="table">
    { list.filter(isSearched(pattern)).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>

export default App;
```

Further Reading:

- Read about the ES6 Object.assign()
- Read about the ES6 array (and object) spread operator