Split Up Components

This lesson explains how to break-down a large component into smaller components.

Now we have one large App component that keeps growing and may eventually become too complex to manage efficiently. We need to split it into smaller, more manageable parts by creating separate components for search input and the items list.

```
class App extends Component {

  ...

  render() {
    const { searchTerm, list } = this.state;
    return (
      <div className="App">
        <Search />
        <Table />
      </div>
    );
  }
}
```

We pass the components properties that they can use themselves. The App component needs to pass the properties managed in the local state and its class methods.

```
class App extends Component {

  ...

  render() {
    const { searchTerm, list } = this.state;
    return (
      <div className="App">
        <Search
          value={searchTerm}
          onChange={this.onSearchChange}
        />
        <Table
          list={list}
          pattern={searchTerm}
          onDismiss={this.onDismiss}
        />
      </div>
    );

  }
}
```

Now we define the components next to the App component, which will be done using JavaScript ES6 by using classes. They render the same elements as before.

The first one is the Search component:

```
class App extends Component {
  ...
}

class Search extends Component {
  render() {
    const { value, onChange } = this.props;
    return (
      <form>
        <input
          type="text"
          value={value}
          onChange={onChange}
        />
      </form>
    );
  }
}
```

The second one is the Table component.

```
...

class Table extends Component {
  render() {
    const { list, pattern, onDismiss } = this.props;
    return (
      <div>
        {list.filter(isSearched(pattern)).map(item =>
          <div key={item.objectID}>
            <span>
              <a href={item.url}>{item.title}</a>
            </span>
            <span>{item.author}</span>
            <span>{item.num_comments}</span>
            <span>{item.points}</span>
            <span>
              <button
                onClick={() => onDismiss(item.objectID)}
                type="button"
              >
                Dismiss
              </button>
            </span>
          </div>

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

Now you have three ES6 class components. Notice the `props` object is accessible via the class instance by using `this`. Props, short for properties, have all the values passed to the components when we used App component. That way, components can pass properties down the component tree.

By extracting these components from the App component, they become reusable. Since components get their values using the `props` object, you can pass different props to your components every time you use them somewhere else.

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

const list = [
  {
    title: 'React',
    url: 'https://reactjs.org/',
    author: 'Jordan Walke',
    num_comments: 3,
    points: 4,
    objectID: 0,
  },
  {
    title: 'Redux',
    url: 'https://redux.js.org/',
    author: 'Dan Abramov, Andrew Clark',
    num_comments: 2,
    points: 5,
    objectID: 1,
  },
];

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

class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      list,
      searchTerm: '',
    };
    this.onSearchChange = this.onSearchChange.bind(this);
    this.onDismiss = this.onDismiss.bind(this);
  }


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

  onDismiss(id) {
    const isNotId = item => item.objectID !== id;
    const updatedList = this.state.list.filter(isNotId);
    this.setState({ list: updatedList });
  }

  render() {
    const { searchTerm, list } = this.state;
    return (
      <div className="App">
        <Search
          value={searchTerm}
          onChange={this.onSearchChange}
        />
        <Table
          list={list}
          pattern={searchTerm}
          onDismiss={this.onDismiss}
        />
      </div>
    );
  }
}

class Search extends Component {
  render() {
    const { value, onChange } = this.props;
    return (
      <form>
        <input
          type="text"
          value={value}
          onChange={onChange}
        />
      </form>
    );
  }
}

class Table extends Component {
  render() {
    const { list, pattern, onDismiss } = this.props;
    return (
      <div>
        {list.filter(isSearched(pattern)).map(item =>
          <div key={item.objectID}>
            <span>
              <a href={item.url}>{item.title}</a>
            </span>
            <span>{item.author}</span>
            <span>{item.num_comments}</span>
            <span>{item.points}</span>
            <span>
              <button
```

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

export default App;
```

Exercises:

- Discover more components that can be split up like the Search and Table components, but wait until we’ve covered more of its concepts before you implement any of them.