Composable Components

In this lesson, you will get to learn about composable components. You will also extend the search functionality that we implemented in the previous lessons.

The `children` prop is used to pass elements to components from above, which are unknown to the component itself but make it possible to compose components together. We'll see how this looks when you pass a text string as a child to the Search component.

```
class App extends Component {

  ...

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

Now the Search component can destructure the `children` property from the `props` object, and specify where it should be displayed.

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

```
          value={value}
          onChange={onChange}
        />
      </form>
    );
  }
}
```

The “Search” text should now be visible next to your input field. When you use the Search component elsewhere, you can use different entities, since it’s not just text that can be passed as children. You can also pass an element, or element trees that can be encapsulated by components, as children. The children property makes it possible to weave components into each other.

```
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}
        >
          Search
        </Search>
        <Table
          list={list}
          pattern={searchTerm}
          onDismiss={this.onDismiss}
        />
      </div>
    );
  }
}

class Search extends Component {
  render() {
    const { value, onChange, children } = this.props;
    return (
      <form>
        {children} <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;
```

Further Reading:

- Read about the composition model of React