Unidirectional Data Flow

Learn how to manipulate a static state and its component.

Now you have a local state in your App component, but haven't manipulated the state yet. The local state is static, so its component is as well. A good way to experience state manipulation is to engage in component interaction.

We will practice this concept by adding a button for each item in the displayed list. The button will read "Dismiss", as its purpose will be to remove an item from the list. In an email client, for example, it would be a useful way to mark some list items as 'read' while keeping the unread items separate.

```
class App extends Component {

  ...

  render() {
    return (
      <div className="App">
        {this.state.list.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={() => this.onDismiss(item.objectID)}
                type="button"
              >
                Dismiss
              </button>
            </span>
          </div>
        )}
      </div>
    );
  }
}
```

The `onDismiss()` class method is not defined yet. We will do it in a moment,

but for now, let's focus on the `onClick` handler of the button element. As you can see, the `onDismiss()` method in the `onClick` handler is enclosed by an arrow function. You can use it to peek at the `objectID` property of the `item` object and identify the item to be dismissed. An alternative way would be to define the function outside of the `onClick` handler and only pass the defined function to it. We will cover handlers more in-depth as we move along.

Note the use of multilines for the button element, and how elements with multiple attributes can get disorganized easily. The button element is used with multilines and indentations to keep it readable. While this practice isn't specific to React development, it is a programming style I recommend for cleanliness and your own peace of mind.

Now we will implement the `onDismiss()` functionality. It takes an id to identify the item to dismiss. The function is bound to the class and thus becomes a class method. That's why you access it with `this.onDismiss()` and not `onDismiss()`. The `this` object is your class instance. In order to define the `onDismiss()` as class method, you have to bind it in the constructor:

```
class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      list,
    };

    this.onDismiss = this.onDismiss.bind(this);
  }

  onDismiss(id) {
    ...
  }

  render() {
    ...
  }
}
```

In the next step, we define its functionality, the business logic, in the class:

```
onDismiss(id) {
  const updatedList = this.state.list.filter(function isNotId(item) {
    return item.objectID !== id;
  });
}
```

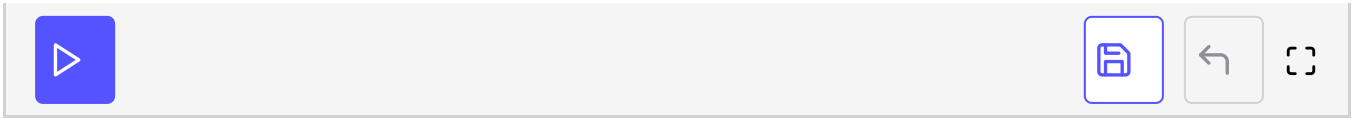
Now we can define what happens inside a class method. Remember, the objective is to remove the item identified by the id from the list and store an updated list to the local state. The updated list will be used in the re-running `render()` method to display it, where the removed item should no longer appear.

You can remove an item from a list using JavaScript's built-in filter functionality, which takes a function as input. The function has access to each value in the list because it iterates over each item, so you can evaluate them based on certain conditions.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const filteredWords = words.filter(function (word) { return word.length > 6; });

console.log(filteredWords);
// expected output: Array ["exuberant", "destruction", "present"]
```



The function returns a new list instead of mutating the old one, and it supports the React convention of using immutable data structures.

```
onDismiss(id) {
  const updatedList = this.state.list.filter(function isNotId(item) {
    return item.objectID !== id;
  });
}
```

In the next step, we extract the function and pass it to the filter function:

```
onDismiss(id) {
  function isNotId(item) {
    return item.objectID !== id;
  }

  const updatedList = this.state.list.filter(isNotId);
}
```

Remember: You can filter more efficiently using a JavaScript ES6 arrow function.

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

You could even inline it again like you did in the `onClick` handler of the button, though it might get less readable:

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

The list removes the clicked item now, but the state hasn't updated yet. Use the `setState()` class method to update the list in the local component state:

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

Run the code again and try the "Dismiss" button. What you experienced is the **unidirectional data flow** of React. An action is triggered in the view layer with `onClick()`, a function or class method modifies the local component state, and then the `render()` method of the component runs again to update the view.

```
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,
  },
];


class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      list,
    };

    this.onDismiss = this.onDismiss.bind(this);
  }

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

  render() {
    return (
      <div className="App">
        { this.state.list.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={() => this.onDismiss(item.objectID)}
                type="button"
              >
                Dismiss
              </button>
            </span>
          </div>
        )}
      </div>
    );
  }
}

export default App;
```

Further Reading:

- Read about the state and lifecycle in React