5 Antworten

Diego Caponera vor 40 Minuten

Controlled Counters

This pattern can be called "multiple controlled components":

- the `Counter` have no internal state;
- they just display information (the `value`), and notify the parent component that the user clicked on them;
- in order to tell the parent on which component we clicked, we need something unique and different for every `Counter`;
- in this case, we passed a `name` prop (exactly like how `<form> does!)
- in the future, it can be an id from a database - it's not important how you call it;
- the other needed ingredient is a **function prop**, to be called when the user clicks;
- in this case, we called the prop `onIncrement` , and we pass `handleIncrement` as value;
- we CALL `onIncrement` from INSIDE the `Counter`, with the `name` as argument;
- see the logs: every time, `name` has a different value! Now we know which component to update.
- in `handleIncrement` we do a bit of spread operator gymnastics (more details come in a moment), et voilá!

Diego Caponera vor 35 Minuten

Adding entries to an array

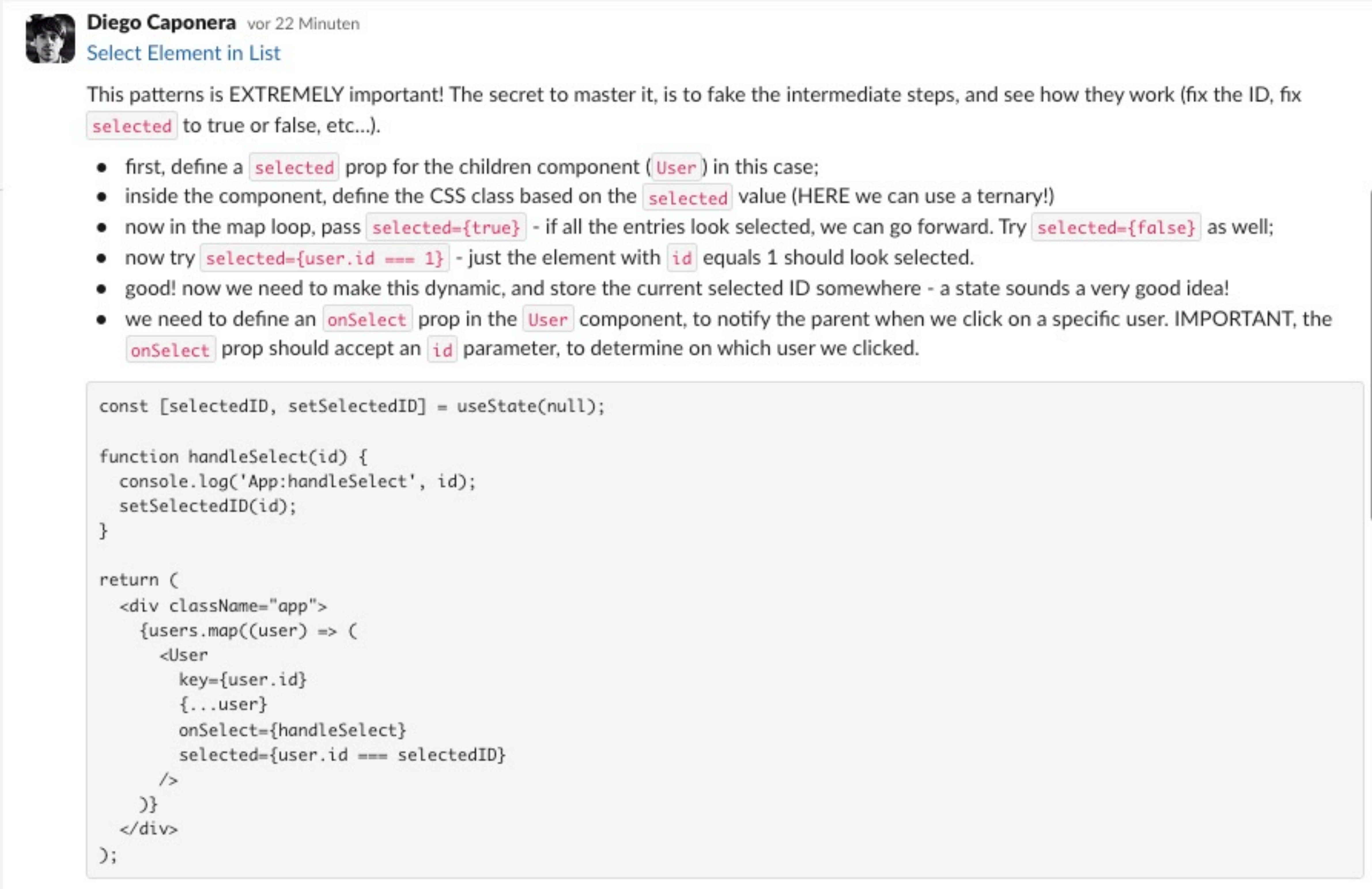
This example works just with primitive values (numbers), but the idea works for objects as well.

If our state is an array and we want to add entries to that, we have to create A NEW array, with the desired content (`Array.prototype.push` does not work with React!).

Without going too much in detail about why it doesn't work for now, let's focus on the syntax:

```
const [numbers, setNumbers] = useState([4, 5, 6]);

function onSubmit(event) {
  event.preventDefault();
  const newNumber = event.target.quantity.value;
  setNumbers(
    [                     // hey setNumbers, I am passing you A NEW array
      ...numbers,         // with whatever was there in numbers already
      newNumber           // and newNumber at the end
    ]
  );
}
```

If you want to add the new entry at the beginning, just do `[newStuff, ….oldStuff]` - look ma, no unshift! (bearbeitet)



Diego Caponera vor 22 Minuten

Select Element in List

This patterns is EXTREMELY important! The secret to master it, is to fake the intermediate steps, and see how they work (fix the ID, fix `selected` to true or false, etc...).

- first, define a `selected` prop for the children component (`User`) in this case;
- inside the component, define the CSS class based on the `selected` value (HERE we can use a ternary!)
- now in the map loop, pass `selected={true}` - if all the entries look selected, we can go forward. Try `selected={false}` as well;
- now try `selected={user.id === 1}` - just the element with `id` equals 1 should look selected.
- good! now we need to make this dynamic, and store the current selected ID somewhere - a state sounds a very good idea!
- we need to define an `onSelect` prop in the `User` component, to notify the parent when we click on a specific user. IMPORTANT, the `onSelect` prop should accept an `id` parameter, to determine on which user we clicked.

```
const [selectedID, setSelectedID] = useState(null);

function handleSelect(id) {
  console.log('App:handleSelect', id);
  setSelectedID(id);
}

return (
  <div className="app">
    {users.map((user) => (
      <User
        key={user.id}
        {...user}
        onSelect={handleSelect}
        selected={user.id === selectedID}
      />
    ))}
  </div>
);
```

Inside the child component:

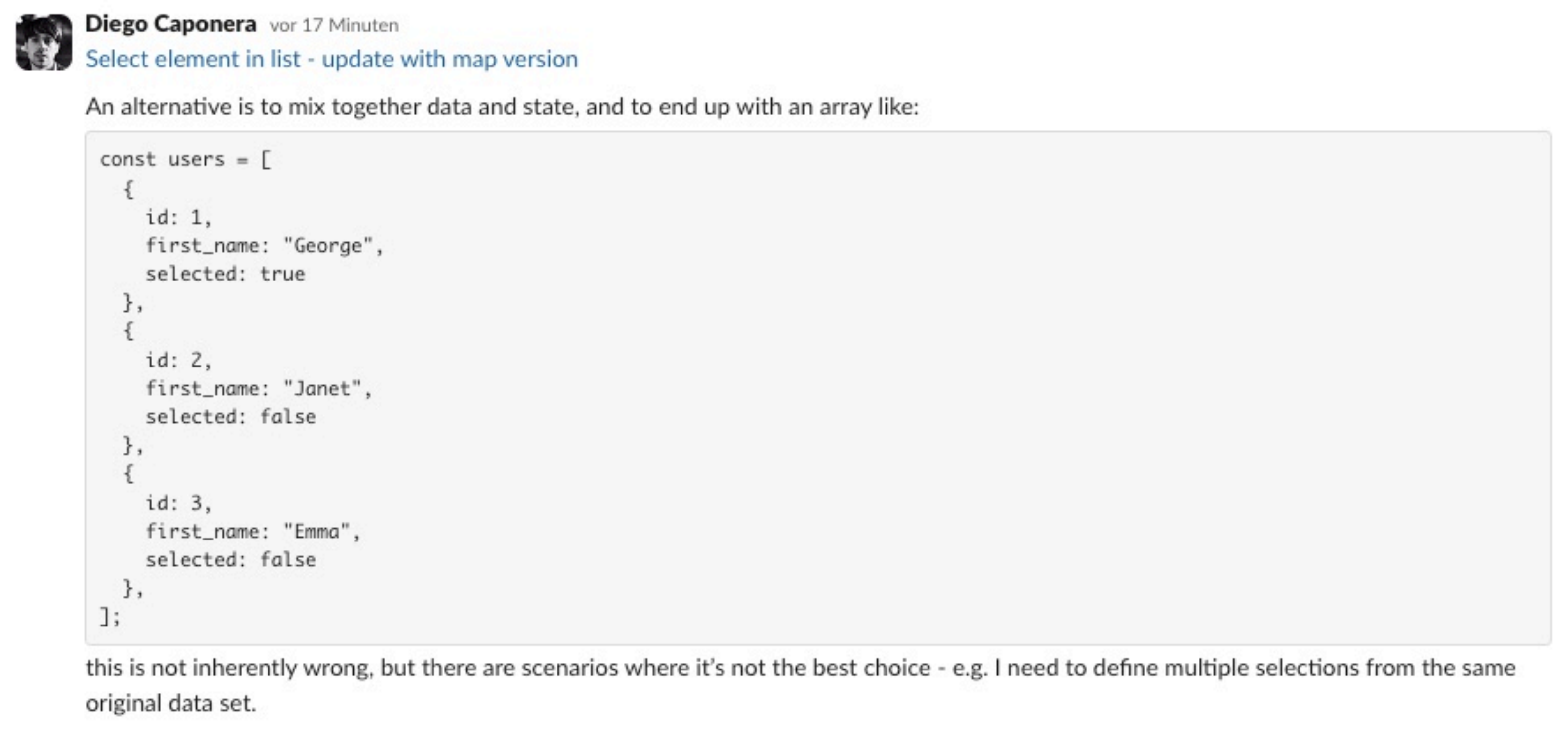
```
function User({ id, onSelect, selected, ... }) {
  function onClick() {
    onSelect(id);
  }

  const className = selected ? 'box selected' : 'box';

  return (
    <div className={className}>
      ...
      <button onClick={onClick}>Select</button>
    </div>
  );
}
```

Again, we need three ingredients to make this work:

- the `selected` prop, for deciding the visual final effect in CSS;
- the `onSelect` function, to notify the parent that we clicked;
- the `id` to pass to `onSelect`, to update the parent `selectedID` with the id of the clicked element.

The "reactive" part is then `selected={user.id === selectedID}` - it determines the used CSS class.

It's extremely important to separate all the processes, that now are rightfully a mess in your head - aim at untangling the mess in the next weeks and don't hesitate to ask as many questions as possible. This is 100% probable interview material!



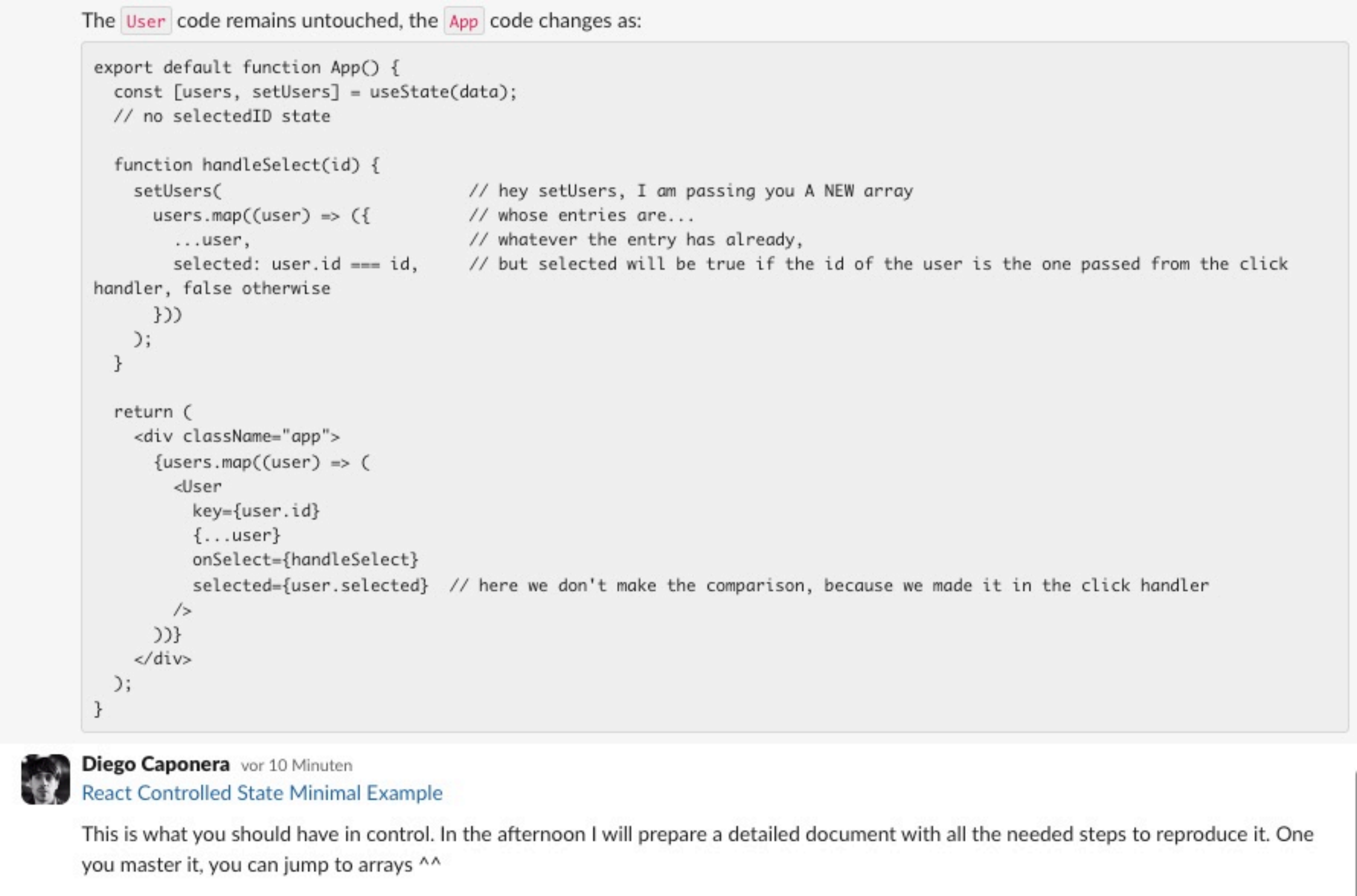
Diego Caponera vor 17 Minuten

Select element in list - update with map version

An alternative is to mix together data and state, and to end up with an array like:

```
const users = [
  {
    id: 1,
    first_name: "George",
    selected: true
  },
  {
    id: 2,
    first_name: "Janet",
    selected: false
  },
  {
    id: 3,
    first_name: "Emma",
    selected: false
  },
];
```

this is not inherently wrong, but there are scenarios where it's not the best choice - e.g. I need to define multiple selections from the same original data set.



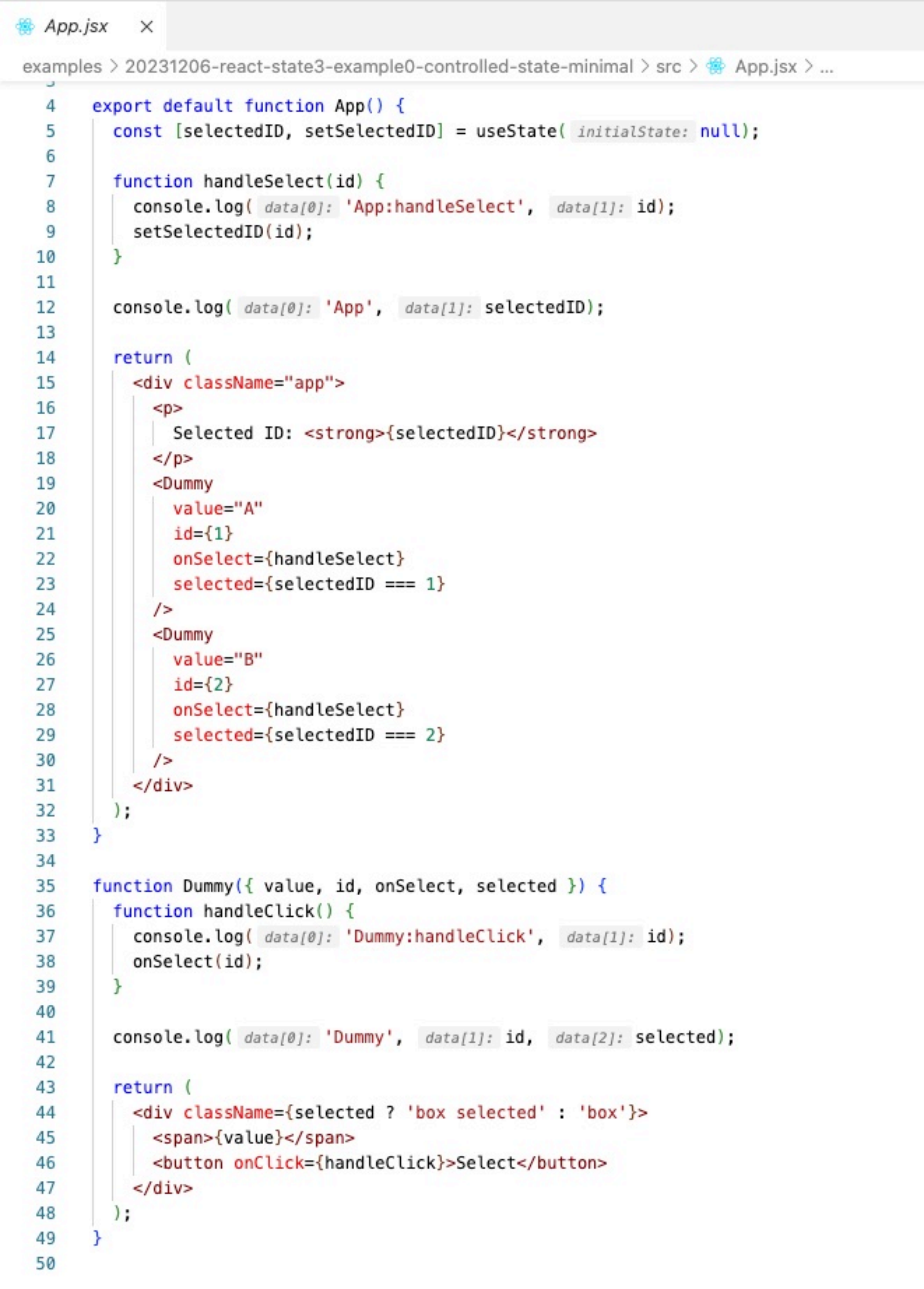
The `User` code remains untouched, the `App` code changes as:

```
export default function App() {
  const [users, setUsers] = useState(data);
  // no selectedID state

  function handleSelect(id) {
    setUsers(                           // hey setUsers, I am passing you A NEW array
      users.map((user) => ({            // whose entries are...
        ...user,                        // whatever the entry has already,
        selected: user.id === id,       // but selected will be true if the id of the user is the one passed from the click
handler, false otherwise
      }))
    );
  }

  return (
    <div className="app">
      {users.map((user) => (
        <User
          key={user.id}
          {...user}
          onSelect={handleSelect}
          selected={user.selected}  // here we don't make the comparison, because we made it in the click handler
        />
      ))}
    </div>
  );
}
```

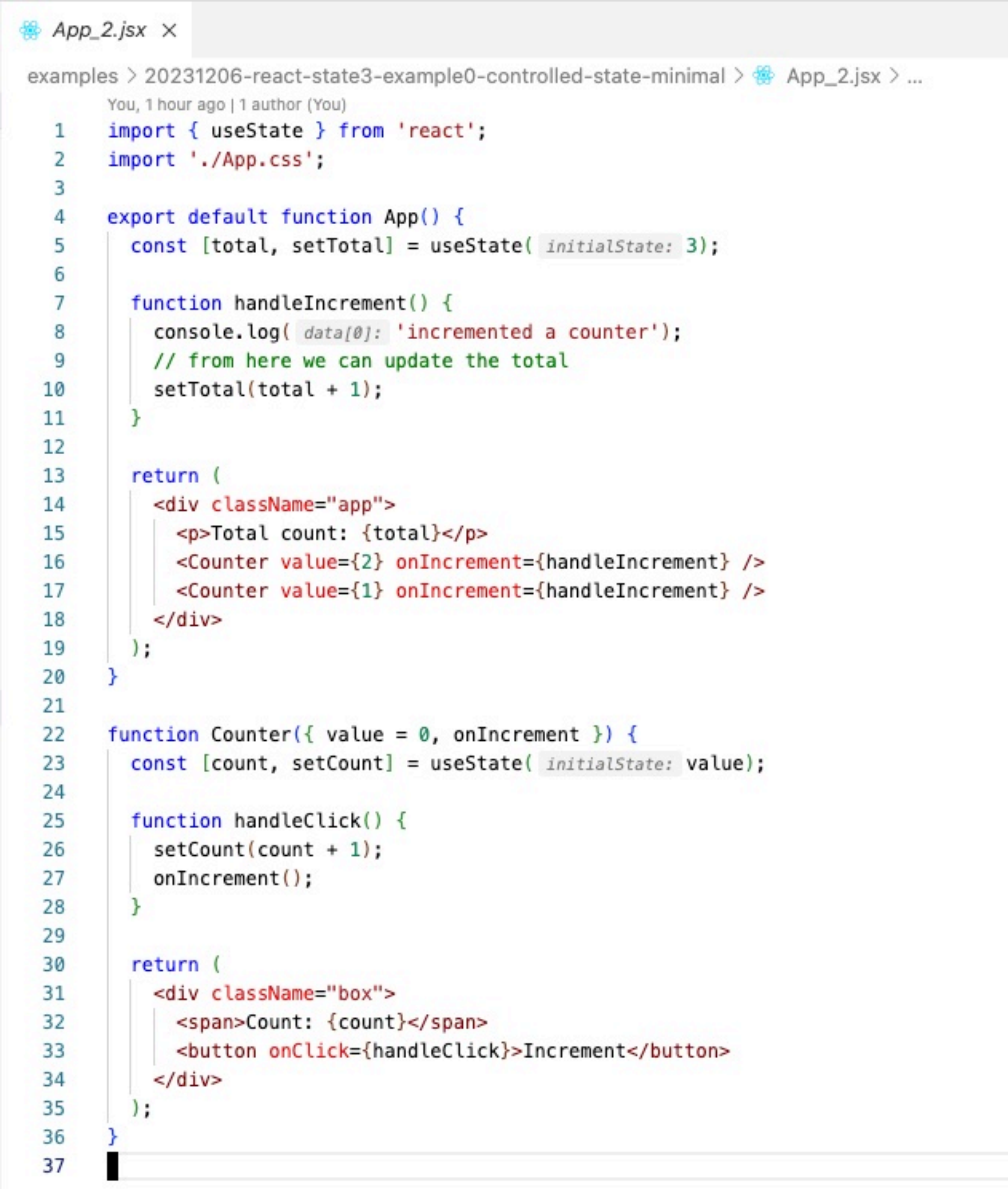
Diego Caponera vor 10 Minuten

React Controlled State Minimal Example

This is what you should have in control. In the afternoon I will prepare a detailed document with all the needed steps to reproduce it. One you master it, you can jump to arrays ^^


App.jsx
examples > 20231206-react-state3-example0-controlled-state-minimal > src > App.jsx > ...

```jsx
export default function App() {
  const [selectedID, setSelectedID] = useState( initialState: null);

  function handleSelect(id) {
    console.log( data[0]: 'App:handleSelect',  data[1]: id);
    setSelectedID(id);
  }

  console.log( data[0]: 'App',  data[1]: selectedID);

  return (
    <div className="app">
      <p>
        Selected ID: <strong>{selectedID}</strong>
      </p>
      <Dummy
        value="A"
        id={1}
        onSelect={handleSelect}
        selected={selectedID === 1}
      />
      <Dummy
        value="B"
        id={2}
        onSelect={handleSelect}
        selected={selectedID === 2}
      />
    </div>
  );
}

function Dummy({ value, id, onSelect, selected }) {
  function handleClick() {
    console.log( data[0]: 'Dummy:handleClick',  data[1]: id);
    onSelect(id);
  }

  console.log( data[0]: 'Dummy',  data[1]: id,  data[2]: selected);

  return (
    <div className={selected ? 'box selected' : 'box'}>
      <span>{value}</span>
      <button onClick={handleClick}>Select</button>
    </div>
  );
}
```

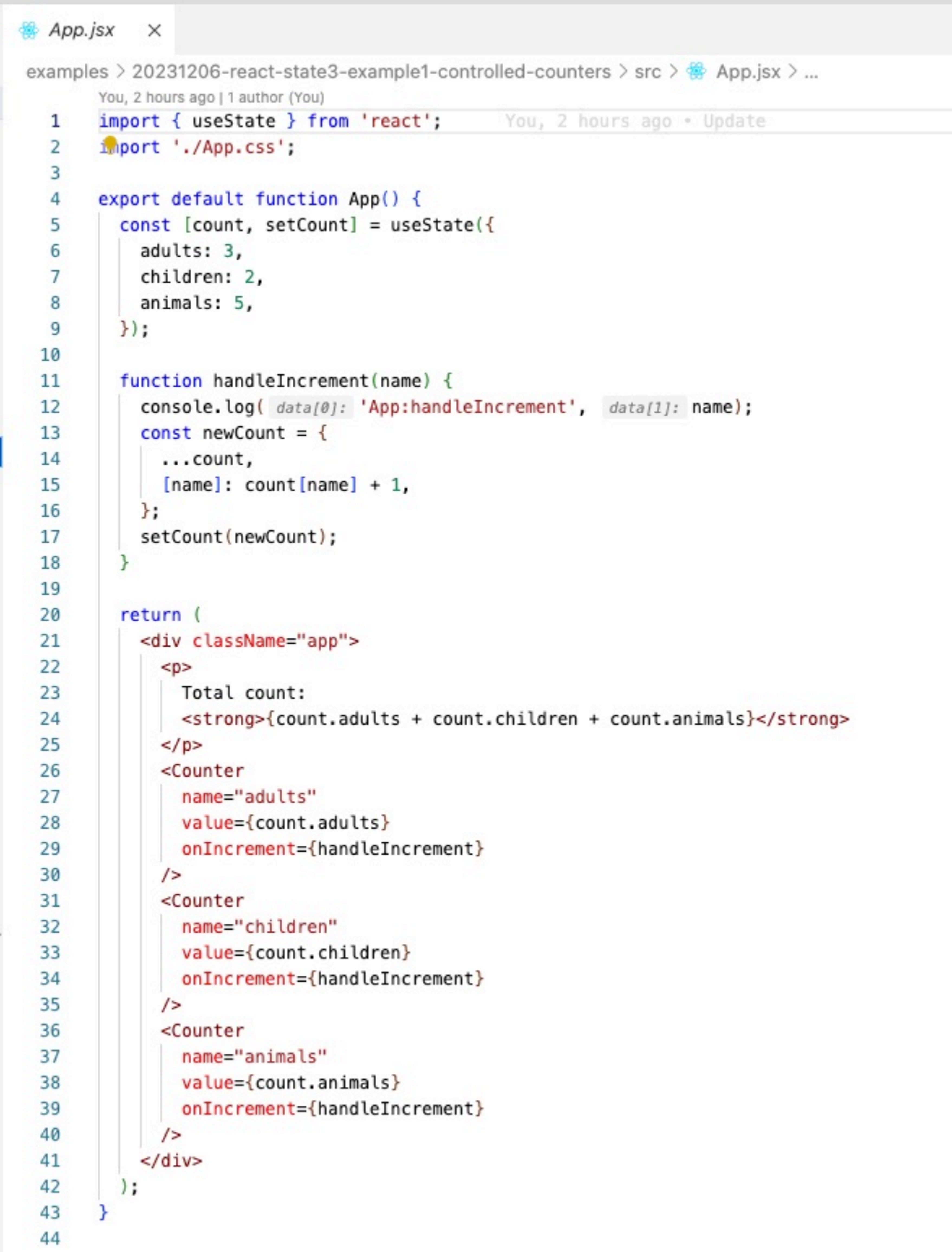


App_2.jsx ×

examples > 20231206-react-state3-example0-controlled-state-minimal > App_2.jsx > ...

You, 1 hour ago | 1 author (You)

```jsx
import { useState } from 'react';
import './App.css';

export default function App() {
  const [total, setTotal] = useState( initialState: 3);

  function handleIncrement() {
    console.log( data[0]: 'incremented a counter');
    // from here we can update the total
    setTotal(total + 1);
  }

  return (
    <div className="app">
      <p>Total count: {total}</p>
      <Counter value={2} onIncrement={handleIncrement} />
      <Counter value={1} onIncrement={handleIncrement} />
    </div>
  );
}

function Counter({ value = 0, onIncrement }) {
  const [count, setCount] = useState( initialState: value);

  function handleClick() {
    setCount(count + 1);
    onIncrement();
  }

  return (
    <div className="box">
      <span>Count: {count}</span>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}
```

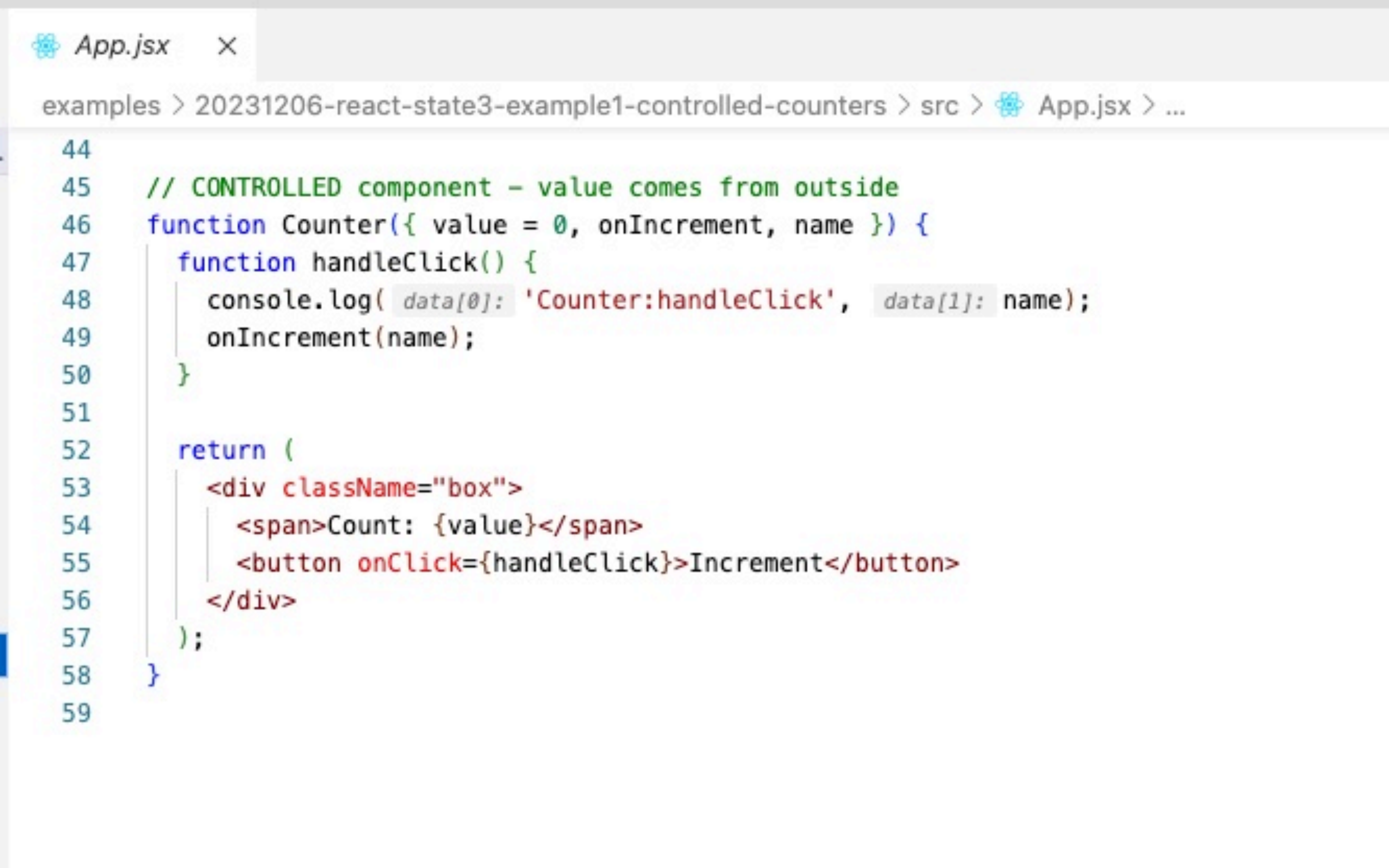


App.jsx ×

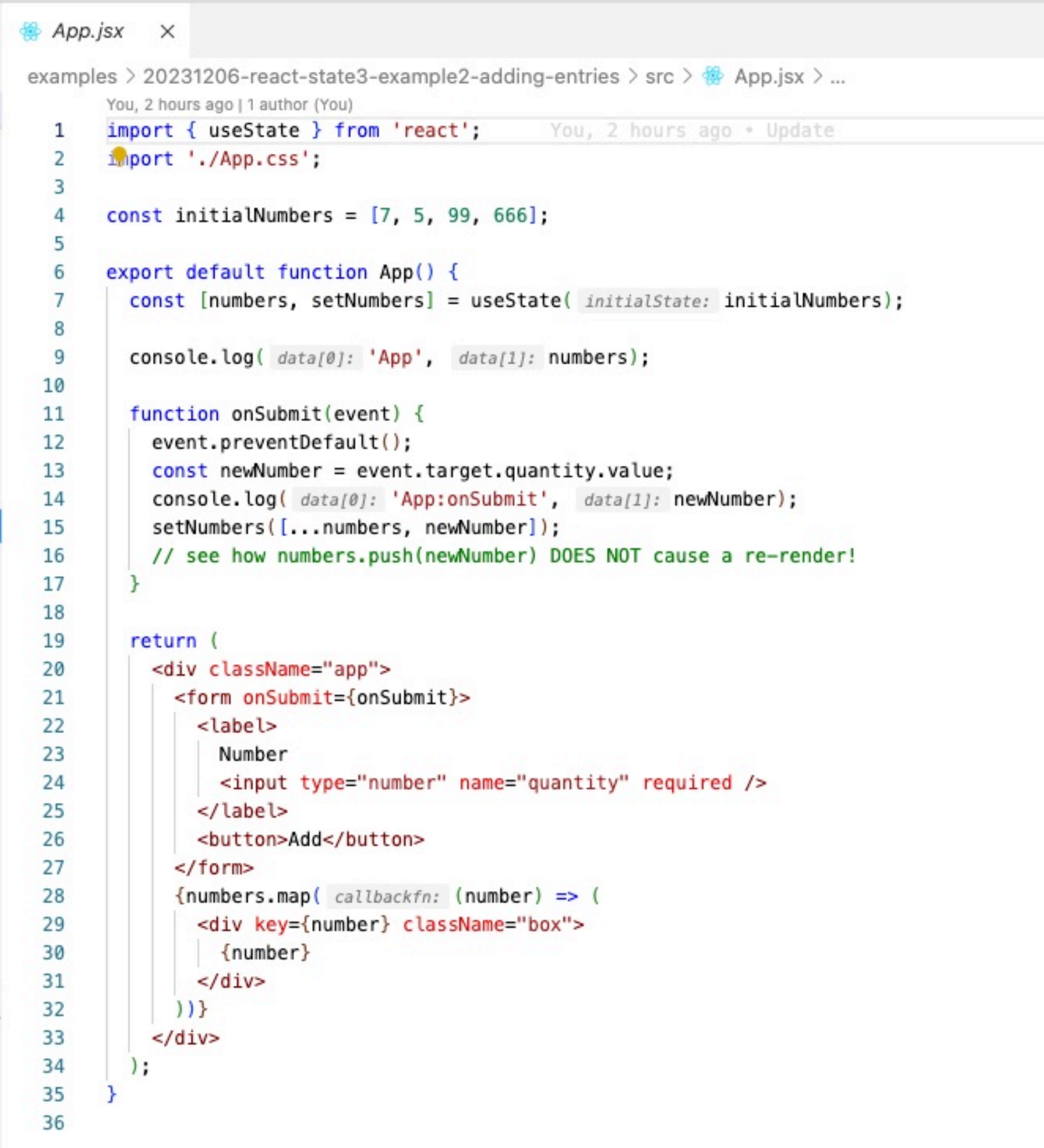
examples > 20231206-react-state3-example1-controlled-counters > src > App.jsx > ...

You, 2 hours ago | 1 author (You)

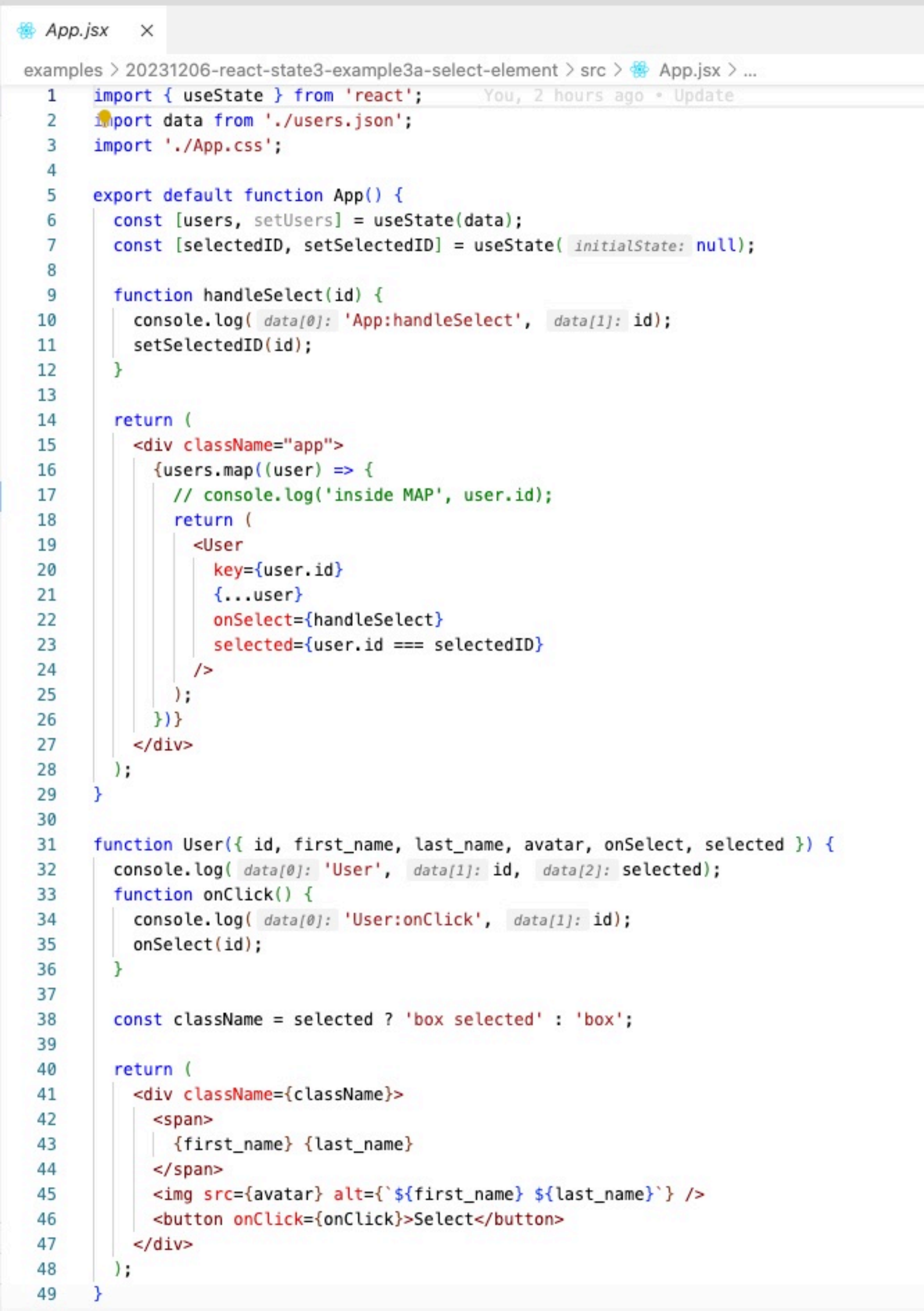
```
import { useState } from 'react';       You, 2 hours ago • Update
import './App.css';

export default function App() {
  const [count, setCount] = useState({
    adults: 3,
    children: 2,
    animals: 5,
  });

  function handleIncrement(name) {
    console.log( data[0]: 'App:handleIncrement',  data[1]: name);
    const newCount = {
      ...count,
      [name]: count[name] + 1,
    };
    setCount(newCount);
  }

  return (
    <div className="app">
      <p>
        Total count:
        <strong>{count.adults + count.children + count.animals}</strong>
      </p>
      <Counter
        name="adults"
        value={count.adults}
        onIncrement={handleIncrement}
      />
      <Counter
        name="children"
        value={count.children}
        onIncrement={handleIncrement}
      />
      <Counter
        name="animals"
        value={count.animals}
        onIncrement={handleIncrement}
      />
    </div>
  );
}
```


App.jsx
examples > 20231206-react-state3-example1-controlled-counters > src > App.jsx > ...

```jsx

// CONTROLLED component - value comes from outside
function Counter({ value = 0, onIncrement, name }) {
  function handleClick() {
    console.log( data[0]: 'Counter:handleClick',  data[1]: name);
    onIncrement(name);
  }

  return (
    <div className="box">
      <span>Count: {value}</span>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}

```

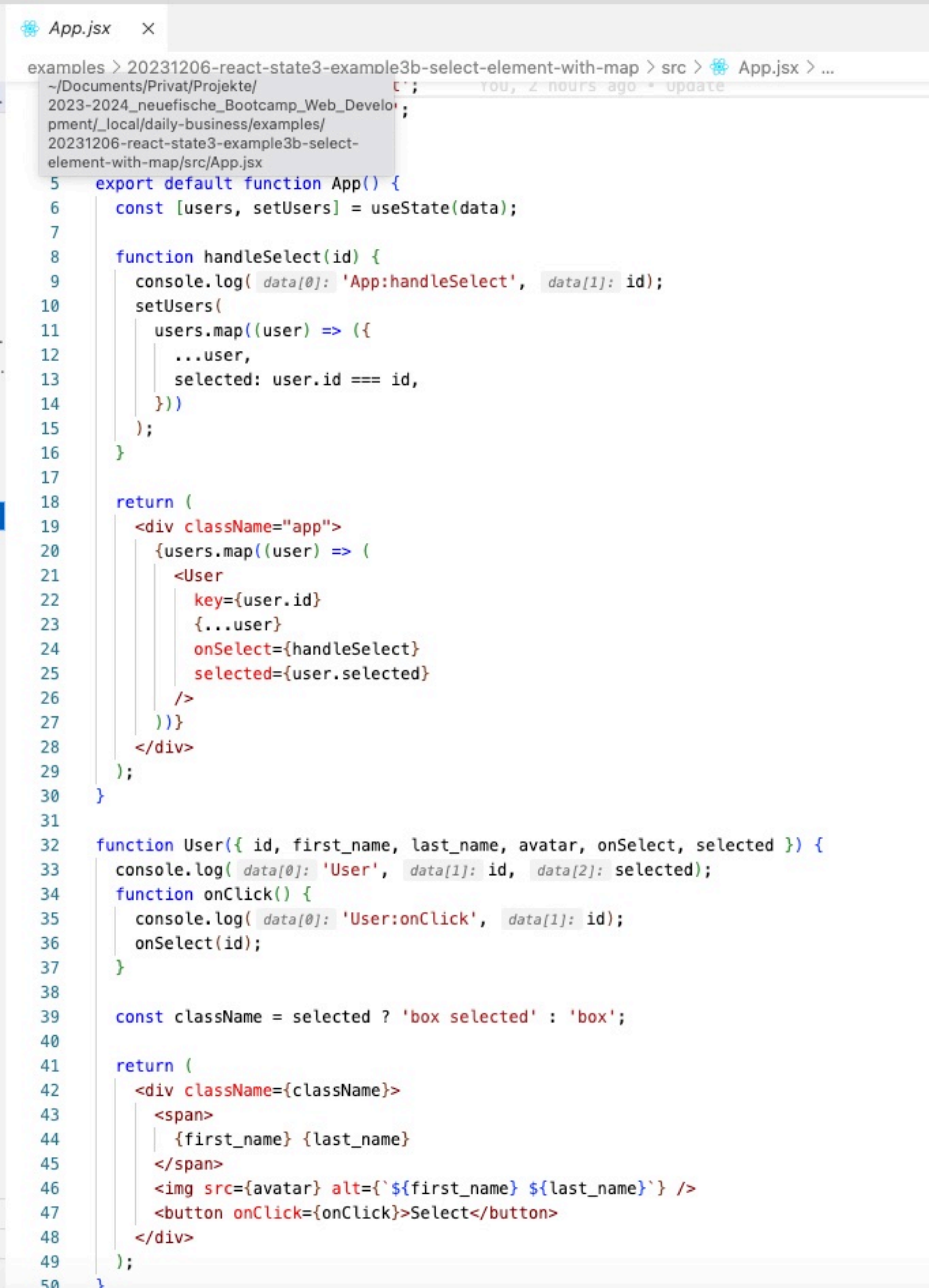

App.jsx
examples > 20231206-react-state3-example2-adding-entries > src > App.jsx > ...
You, 2 hours ago | 1 author (You)
You, 2 hours ago • Update

```jsx
import { useState } from 'react';
import './App.css';

const initialNumbers = [7, 5, 99, 666];

export default function App() {
  const [numbers, setNumbers] = useState( initialState: initialNumbers);

  console.log( data[0]: 'App',  data[1]: numbers);

  function onSubmit(event) {
    event.preventDefault();
    const newNumber = event.target.quantity.value;
    console.log( data[0]: 'App:onSubmit',  data[1]: newNumber);
    setNumbers([...numbers, newNumber]);
    // see how numbers.push(newNumber) DOES NOT cause a re-render!
  }

  return (
    <div className="app">
      <form onSubmit={onSubmit}>
        <label>
          Number
          <input type="number" name="quantity" required />
        </label>
        <button>Add</button>
      </form>
      {numbers.map( callbackfn: (number) => (
        <div key={number} className="box">
          {number}
        </div>
      ))}
    </div>
  );
}

```


App.jsx ×

examples > 20231206-react-state3-example3a-select-element > src > App.jsx > ...

```jsx
import { useState } from 'react';    You, 2 hours ago • Update
import data from './users.json';
import './App.css';

export default function App() {
  const [users, setUsers] = useState(data);
  const [selectedID, setSelectedID] = useState( initialState: null);

  function handleSelect(id) {
    console.log( data[0]: 'App:handleSelect',  data[1]: id);
    setSelectedID(id);
  }

  return (
    <div className="app">
      {users.map((user) => {
        // console.log('inside MAP', user.id);
        return (
          <User
            key={user.id}
            {...user}
            onSelect={handleSelect}
            selected={user.id === selectedID}
          />
        );
      })}
    </div>
  );
}

function User({ id, first_name, last_name, avatar, onSelect, selected }) {
  console.log( data[0]: 'User',  data[1]: id,  data[2]: selected);
  function onClick() {
    console.log( data[0]: 'User:onClick',  data[1]: id);
    onSelect(id);
  }

  const className = selected ? 'box selected' : 'box';

  return (
    <div className={className}>
      <span>
        {first_name} {last_name}
      </span>
      <img src={avatar} alt={`${first_name} ${last_name}`} />
      <button onClick={onClick}>Select</button>
    </div>
  );
}
```

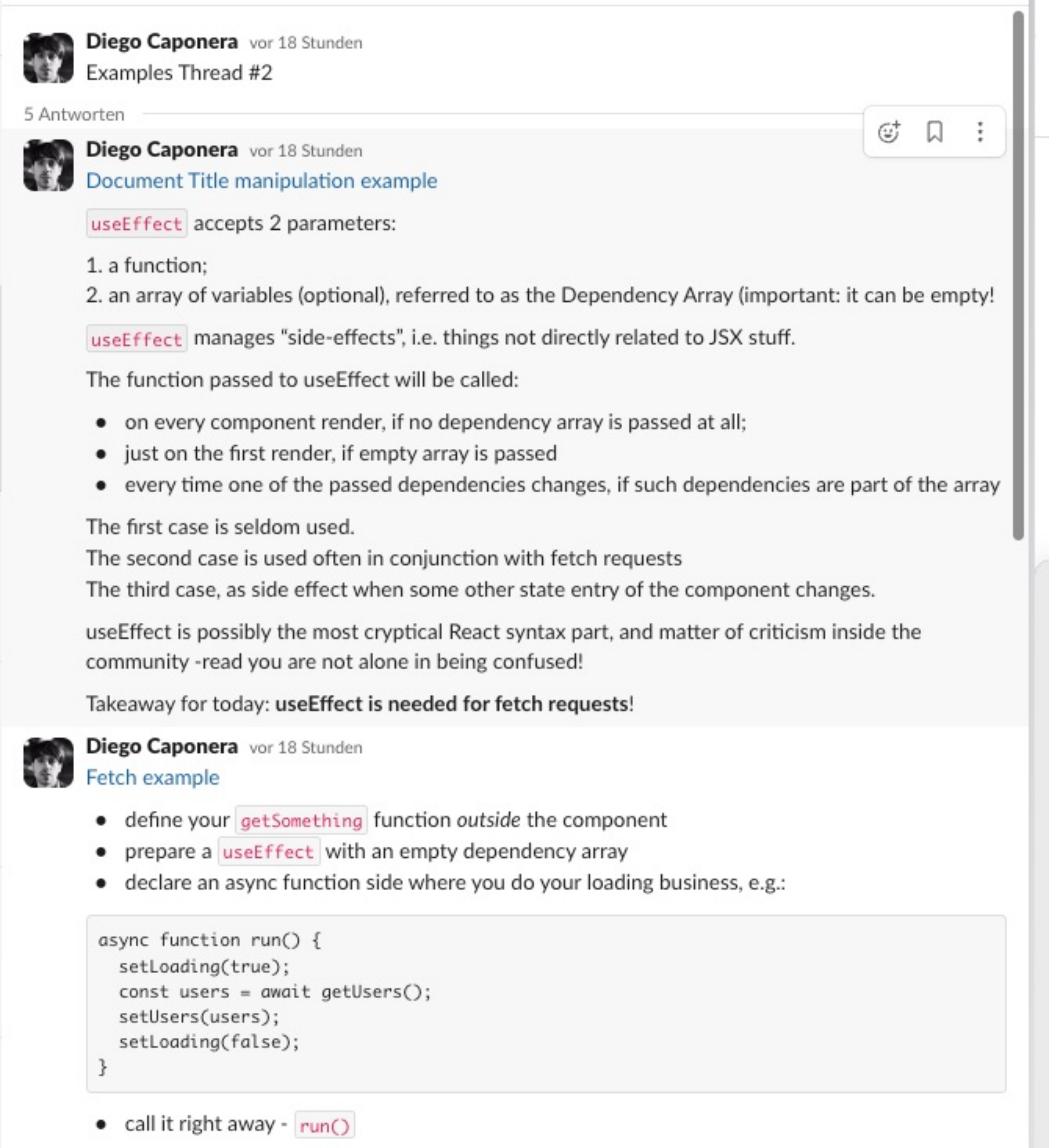


App.jsx

examples > 20231206-react-state3-example3b-select-element-with-map > src > App.jsx > ...

~/Documents/Privat/Projekte/
2023-2024_neuefische_Bootcamp_Web_Develo
pment/_local/daily-business/examples/
20231206-react-state3-example3b-select-
element-with-map/src/App.jsx

```
t';        You, 2 hours ago • Update
;

export default function App() {
  const [users, setUsers] = useState(data);

  function handleSelect(id) {
    console.log( data[0]: 'App:handleSelect',  data[1]: id);
    setUsers(
      users.map((user) => ({
        ...user,
        selected: user.id === id,
      }))
    );
  }

  return (
    <div className="app">
      {users.map((user) => (
        <User
          key={user.id}
          {...user}
          onSelect={handleSelect}
          selected={user.selected}
        />
      ))}
    </div>
  );
}

function User({ id, first_name, last_name, avatar, onSelect, selected }) {
  console.log( data[0]: 'User',  data[1]: id,  data[2]: selected);
  function onClick() {
    console.log( data[0]: 'User:onClick',  data[1]: id);
    onSelect(id);
  }

  const className = selected ? 'box selected' : 'box';

  return (
    <div className={className}>
      <span>
        {first_name} {last_name}
      </span>
      <img src={avatar} alt={`${first_name} ${last_name}`} />
      <button onClick={onClick}>Select</button>
    </div>
  );
}
```



Diego Caponera vor 18 Stunden

Examples Thread #2

5 Antworten

Diego Caponera vor 18 Stunden

Document Title manipulation example

`useEffect` accepts 2 parameters:

1. a function;
2. an array of variables (optional), referred to as the Dependency Array (important: it can be empty!

`useEffect` manages "side-effects", i.e. things not directly related to JSX stuff.

The function passed to useEffect will be called:

- on every component render, if no dependency array is passed at all;
- just on the first render, if empty array is passed
- every time one of the passed dependencies changes, if such dependencies are part of the array

The first case is seldom used.
The second case is used often in conjunction with fetch requests
The third case, as side effect when some other state entry of the component changes.

useEffect is possibly the most cryptical React syntax part, and matter of criticism inside the community -read you are not alone in being confused!

Takeaway for today: **useEffect is needed for fetch requests**!

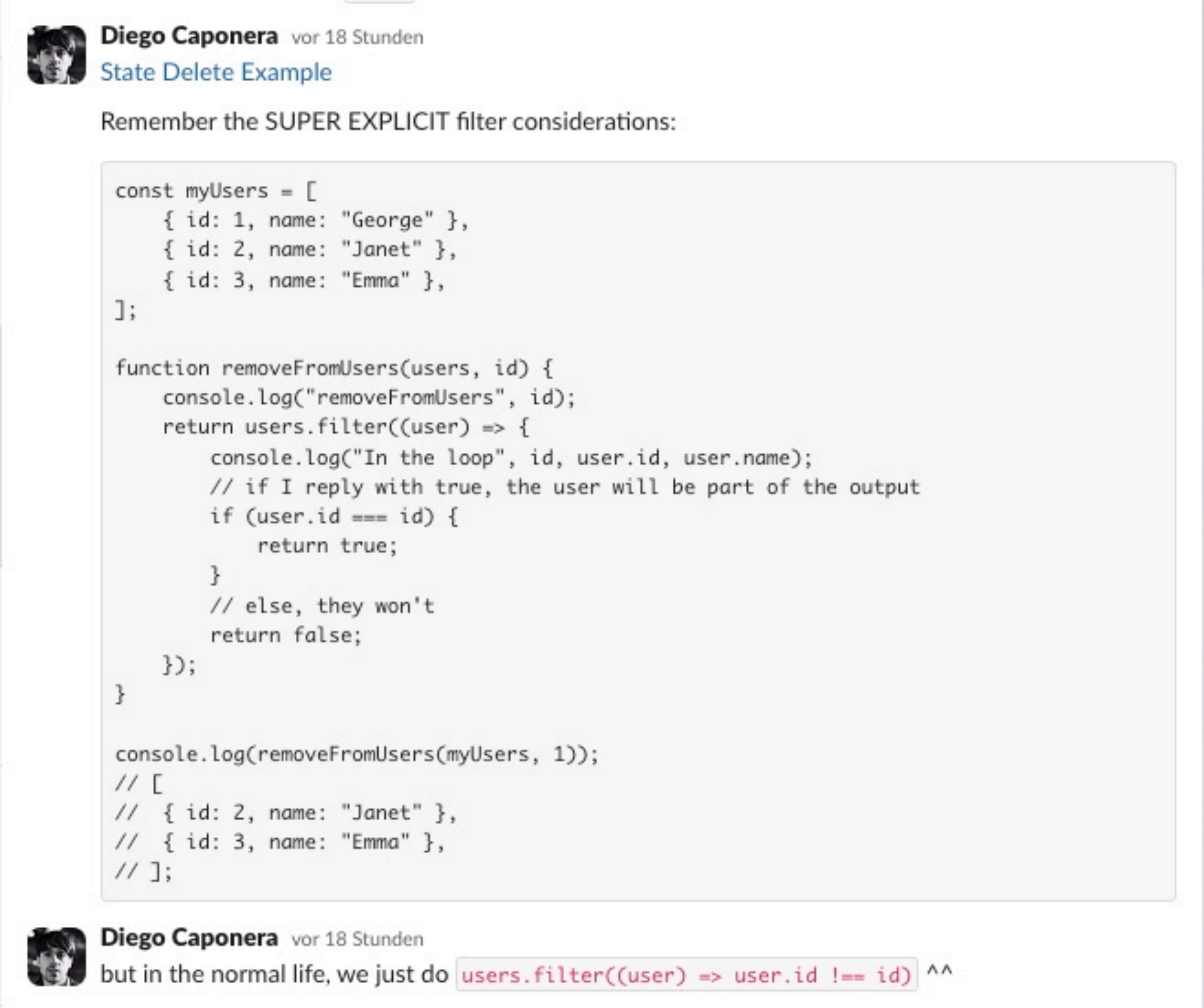
Diego Caponera vor 18 Stunden

Fetch example

- define your `getSomething` function *outside* the component
- prepare a `useEffect` with an empty dependency array
- declare an async function side where you do your loading business, e.g.:

```
async function run() {
  setLoading(true);
  const users = await getUsers();
  setUsers(users);
  setLoading(false);
}
```

- call it right away - `run()`



Diego Caponera vor 18 Stunden

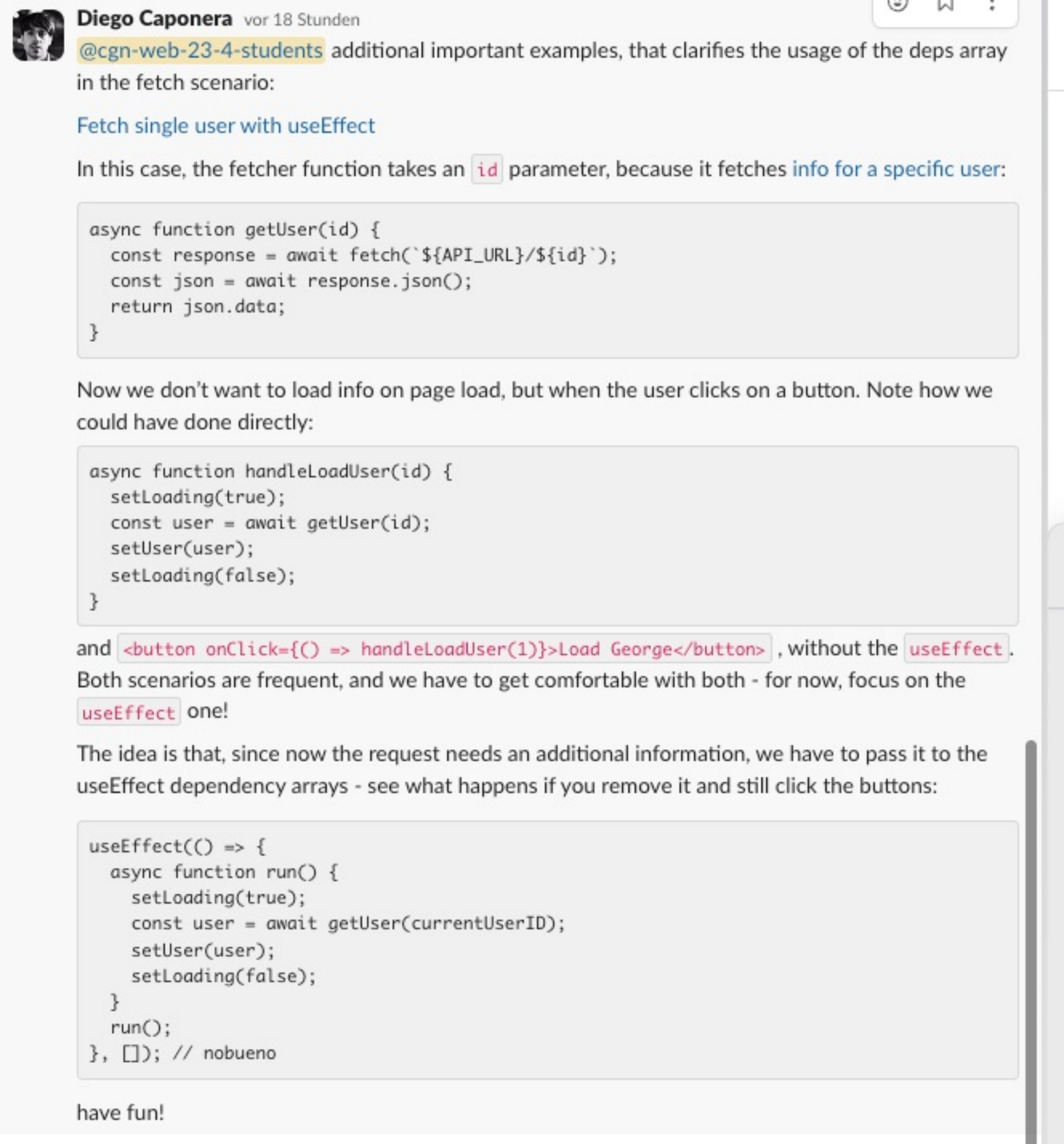
State Delete Example

Remember the SUPER EXPLICIT filter considerations:

```
const myUsers = [
    { id: 1, name: "George" },
    { id: 2, name: "Janet" },
    { id: 3, name: "Emma" },
];

function removeFromUsers(users, id) {
    console.log("removeFromUsers", id);
    return users.filter((user) => {
        console.log("In the loop", id, user.id, user.name);
        // if I reply with true, the user will be part of the output
        if (user.id === id) {
            return true;
        }
        // else, they won't
        return false;
    });
}

console.log(removeFromUsers(myUsers, 1));
// [
//  { id: 2, name: "Janet" },
//  { id: 3, name: "Emma" },
// ];
```

Diego Caponera vor 18 Stunden

but in the normal life, we just do `users.filter((user) => user.id !== id)` ^^



Diego Caponera vor 18 Stunden

@cgn-web-23-4-students additional important examples, that clarifies the usage of the deps array in the fetch scenario:

Fetch single user with useEffect

In this case, the fetcher function takes an `id` parameter, because it fetches info for a specific user:

```
async function getUser(id) {
  const response = await fetch(`${API_URL}/${id}`);
  const json = await response.json();
  return json.data;
}
```

Now we don't want to load info on page load, but when the user clicks on a button. Note how we could have done directly:

```
async function handleLoadUser(id) {
  setLoading(true);
  const user = await getUser(id);
  setUser(user);
  setLoading(false);
}
```

and `<button onClick={() => handleLoadUser(1)}>Load George</button>` , without the `useEffect`. Both scenarios are frequent, and we have to get comfortable with both - for now, focus on the `useEffect` one!

The idea is that, since now the request needs an additional information, we have to pass it to the useEffect dependency arrays - see what happens if you remove it and still click the buttons:

```
useEffect(() => {
  async function run() {
    setLoading(true);
    const user = await getUser(currentUserID);
    setUser(user);
    setLoading(false);
  }
  run();
}, []); // nobueno
```

have fun!