Lecture 12

React.js Wrap-Up

Contents

- React Hooks
  - Managing component state with `useState`
  - Executing side-effects with `useEffect`
- Sharing states amongst different components
- Connecting React front-end with a back-end
  - Example: fetching an Express.js backend API

Managing Component’s State

Managing State for Class Components

- Class component's properties should be kept in an object called `state`
- Example: add the `color` property & use it in `render()`

```
class Car extends React.Component {
    constructor() {
        super();
        this.state = { color: "red" };
    }
    render() {
        return <h2>I am a {this.state.color} Car!</h2>;
    }
}
```

- Refer to the state object anywhere in the component with `this.state`

Modifying the state

- Whenever the state changes, the component should be re-rendered
- We achieve this with the `this.setState()` method

```
class Car extends React.Component {
    constructor(props) {
        super(props);
        this.state = { color: props.color };
    }
    changeColor = () => { this.setState({ color: "blue" }); }
    render() {
        return (<div>
            <h2>I'm a {this.state.color} car</h2>
            <button onClick={this.changeColor}>Change color</button>
        </div>);
    }
}
```

Some notes on props & state

- `props` are read-only
- Always pass `props` to a call to `super()` in the constructor
- You should clone the `state` object when using setState()
  - Using the spread operator
  - Using `JSON.stringify()` and `JSON.parse()`

```
const food = { favorite: 'beef', notfavorite: 'steak' };
// shallow clone only
const cloneFood = { ...food };

// can create deep clone
const cloneFood = JSON.parse(JSON.stringify(food));
```

Function component update example (not gonna work)

```
function Board() {
    let status = 'Hi!';
    const clickHandler = () => {
        status = 'Updated!';
    }
    return (
        <div className="info">
            <div className="status">
                {status}
            </div>
            <button onClick={clickHandler}>Click me</button>
        </div>
    )
}
```

What are React Hooks?

- Hooks allow function components to have access to state and other React features
  - Hooks make function components more powerful
  - Function components and Hooks can replace class components
- Hook rules:
  - Hooks can only be called inside React function components
  - Hooks can only be called at the top level of a component
  - Hooks cannot be conditional (* cannot put hooks under if-else)

Managing State for Function Components

- We use a hook called `useState`

```
import { useState } from "react";
```

- Initialize state at the top of the function component:

```
function Car(props) {
    const [color, setColor] = useState(props.color);
}
```

- In this example, `color` is the current state and `setColor` is the function that is used to update the state.
- When the component is re-rendered, the state is not reset to the initial value.
- Use the state just like a local variable:

```
return <h2>I'm a {color} car</h2>
```

Function component update with `useState` hook (works)

```
import React, { useState } from 'react';

function Board() {
    const [status, updateStatus] = useState('Old value');
    const clickHandler = () => {
        updateStatus('Updated!');
    }
    return (
        <div className="info">
            <div className="status">
                {status}
            </div>
            <button onClick={clickHandler}>Click me</button>
        </div>
    )
}
```

Modifying state on event

```
function Car(props) {
    const [color, setColor] = useState(props.color);
    return (
        <div>
            <h2>I'm a {color} car</h2>
            <button onClick={() => setColor("blue")}>
                Change color
            </button>
        </div>
    );
}
```

- The component will re-render if the changed state is required for rendering the component.

Using multiple state values

- We can create multiple state hooks:

```
function Car() {
    const [brand, setBrand] = useState("Ford");
    const [year, setYear] = useState("1964");
    const [color, setColor] = useState("red");
    return <p>It is a {color} {brand} from {year}.</p>;
}
```

- Or create a single hook that holds an object:

```
const [car, setCar] = useState({
    brand: "Ford",
    year: "1964",
    color: "red"
});
return <p>It is a {car.color} {car.brand} from {car.year}.</p>;
```

Updating Objects and Arrays in State

- What if we only want to update the `color` of our `Car`?

```
function Car() {
    const [car, setCar] = useState({
        brand: "Ford",
        year: "1964",
        color: "red"
    });
    const updateColor = () => {
        setCar({ ...car, color: "blue" });
    }
    return <div>
        <p>It is a {car.color} {car.brand} from {car.year}.</p>
        <button onClick={updateColor}>Change color</button>
    </div>;
}
```

More about `useState` in React

- State is created and managed separately for each Component instance.
  - Different instances of the same Component have different states
- Consider this example:

```
const [status, updateStatus] = useState('First value');
```

  - `const` is used although we plan to update the value later
  - **Reason:** status isn't modified directly (with the = sign)
  - When `updateStatus` is called, React will eventually re-load the Component (which means re-calling the Component function)

Example: counting button

- This button counts the number of clicks on itself:

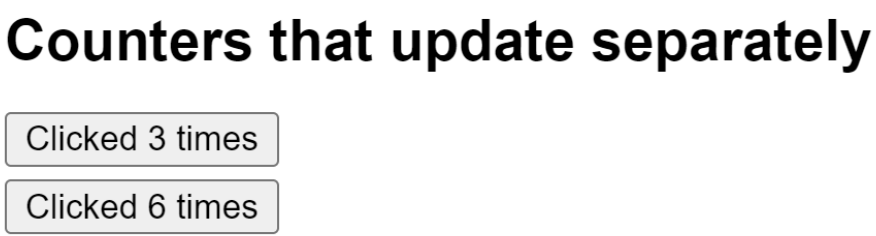
```
function MyButton() {
    const [count, setCount] = useState(0);

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

    return (
        <button onClick={handleClick}>
            Clicked {count} times
        </button>
    );
}
```

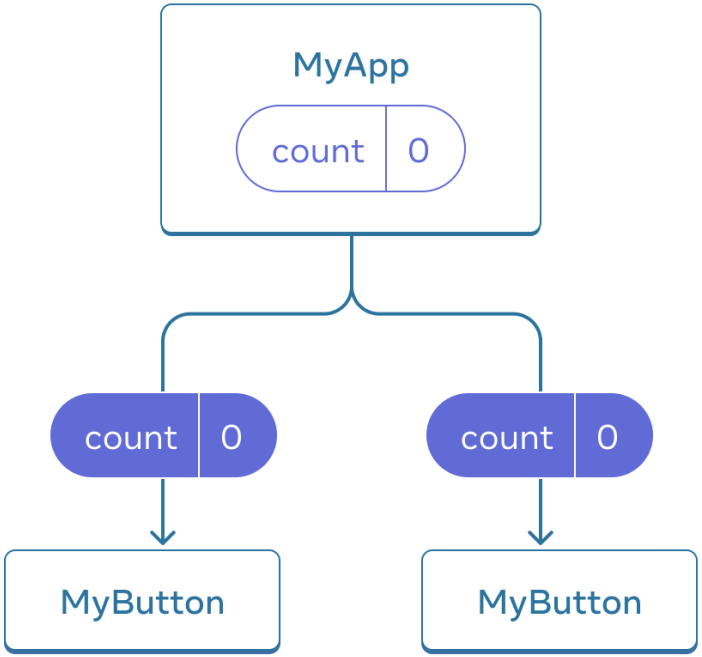
Each component has it own state

```
export default function MyApp() {
    return (
        <div>
            <h1>Counters that update separately</h1>
            <MyButton />
            <MyButton />
        </div>
    );
}
```

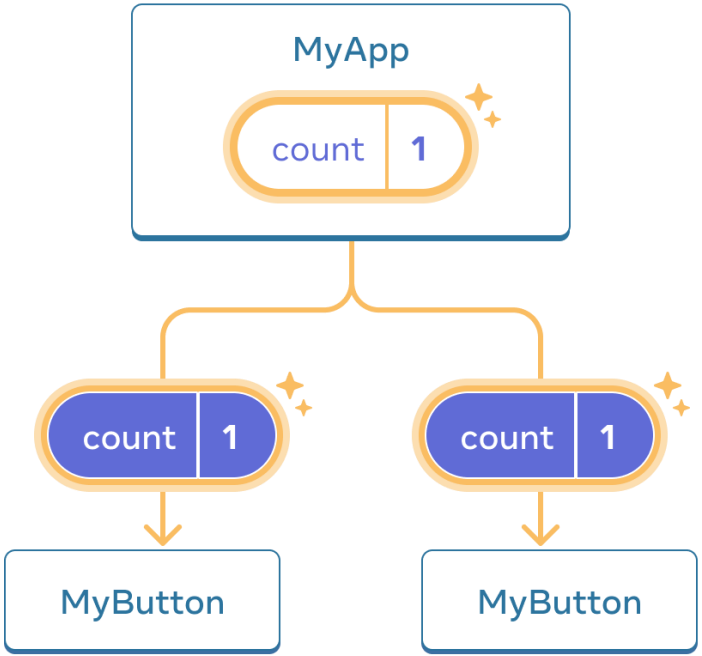


Sharing state across components

- In React, data only flows from parent component to child components, never in the opposite direction.



versus



Step 1

- Move both the state and state-updating function to the parent component:

```
export default function MyApp() {
    const [count, setCount] = useState(0);

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

    return (
        <div>
            <h1>Counters that update together</h1>
            <MyButton count={count} onClick={handleClick} />
            <MyButton count={count} onClick={handleClick} />
        </div>
    );
}
```

Step 2

- Receive the state and the state-updating function as props in child component

```
function MyButton({ count, onClick }) {
    return (
        <button onClick={onClick}>
            Clicked {count} times
        </button>
    );
}
```

The `useEffect` Hook

- The `useEffect` Hook allows you to perform side effects in your components.
  - Examples of side effects: fetching data, directly updating the DOM, and timers...
- `useEffect` accepts two arguments (the 2nd is optional)

```
useEffect(<function>, <dependency>)
```

`useEffect` hook `timer` example

```
import React, { useState, useEffect } from 'react';

function Timer() {
    const [count, setCount] = useState(0);

    useEffect(() => {
        setTimeout(() => {
            setCount((count) => count + 1);
        }, 1000);
    });

    return <h1>I've been rendered {count} times!</h1>;
}
```

Controlling when `useEffect` executes

- No dependency passed:

```
useEffect(() => {
    // Runs with every render
});
```

- An empty array:

```
useEffect(() => {
    // Runs only on the first render
}, []);
```

- Props or state variables:

```
useEffect(() => {
    // Runs on the first render
    // And any time any dependency value changes
}, [props.something, someStateVar]);
```

Connecting Front-end and Back-end

Incorrect username and/or password

Username: admin

Password: ••••••

Login

Login successfully

Username: admin

Password: ••••••

Login

- This is a simple example of the Login feature from a React front-end, powered by an Express.js back-end.

Back-end code

```
app.post('/login', async (req, res) => {
    let u = req.body.username;
    let p = req.body.password;
    let sql = "SELECT * FROM users WHERE username = ? AND password = ?";
    let [rows] = await db.query(sql, [u, p]);
    if (rows.length > 0) {
        res.json({ success: true, user: rows[0] });
    } else {
        res.json({
            success: false,
            message: "Incorrect username and/or password"
        });
    }
});

app.listen(8000, () => console.log('Listening to port 8000...'));
```

Enabling CORS on back-end

- **Problem:** React front-end runs on http://localhost:3000/ and Express back-end runs on http://localhost:8000/
  (which are basically 2 different websites/hosts)
- To allow `fetch` requests from the front-end to back-end, CORS policy need to be set to allow from all origins.
- By default, just using `cors` will allow access from all origins.
- First, install & import the `cors` module:

```
const cors = require('cors');
```

- Secondly, use the cors middleware on your Express app:

```
app.use(cors());
```

Front-end `Login` component code

- The following is the JSX of the `Login` component:

```
export default function Login() {
    // code omitted
    return (
        <div>
            {msg ? <p>{msg}</p> : null}
            <p>Username: <input type="text" onChange={handleUserChange} /></p>
            <p>Password: <input type="password" onChange={handlePassChange} /></p>
            <p><button onClick={handleLogin}>Login</button></p>
        </div>
    );
}
```

Front-end `Login` component code

- At the top of the function component, some states are created using `useState` hook:

```
export default function Login() {
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");
    const [msg, setMsg] = useState(null);
    const handleUserChange = (e) => setUsername(e.target.value);
    const handlePassChange = (e) => setPassword(e.target.value);
```

Front-end `Login` component code

- A function to handle/process the login action (it utilizes `fetch`):

```
export default function Login() {
    // declaring states
    const handleLogin = () => {
        fetch("http://localhost:8000/login", {
            method: 'post',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username: username, password: password })
        }).then(res => res.json())
          .then(obj => {
              if (obj.success) {
                  setMsg('Login successfully');
              } else {
                  setMsg(obj.message);
              }
          });
    };
    return /* Component's JSX */;
}
```