Advanced React Components Workshop

Maurice de Beijer - @mauricedb
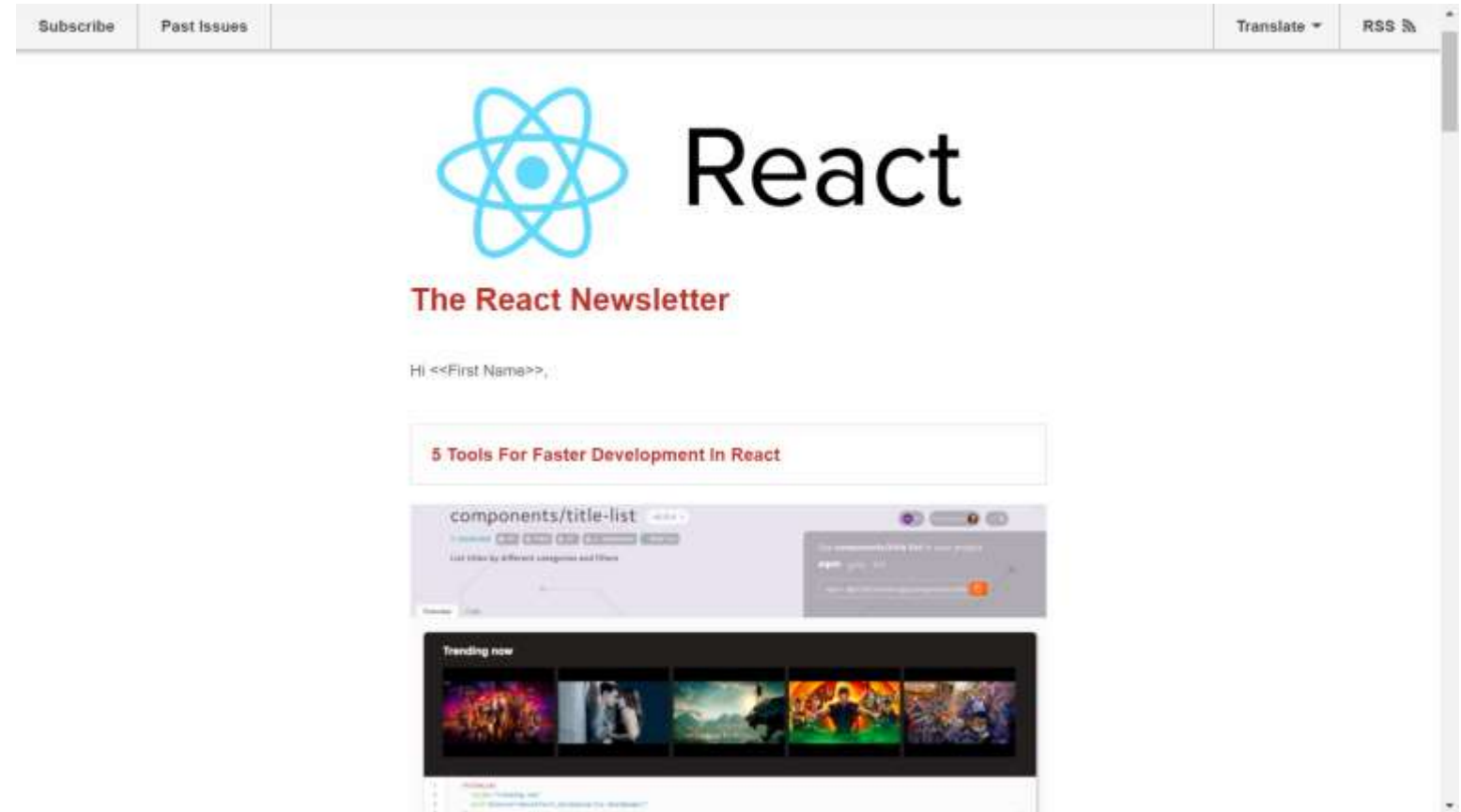
- Maurice de Beijer

- The Problem Solver

- Microsoft MVP

- Freelance developer/instructor

- Twitter: @mauricedb

- Web: http://www.TheProblemSolver.nl

- E-mail: maurice.de.beijer@gmail.com

Skillshare

The React Newsletter

Workshop goal

- Create better React components
- Make them faster and more reliable

- Topics
  1. Controlled versus Uncontrolled components
  2. Fragments
  3. Error Handling
  4. Higher Order Components versus Render Props
  5. New Context API
  6. Pofiler API
  7. PureComponent versus getDerivedStateFromProps()
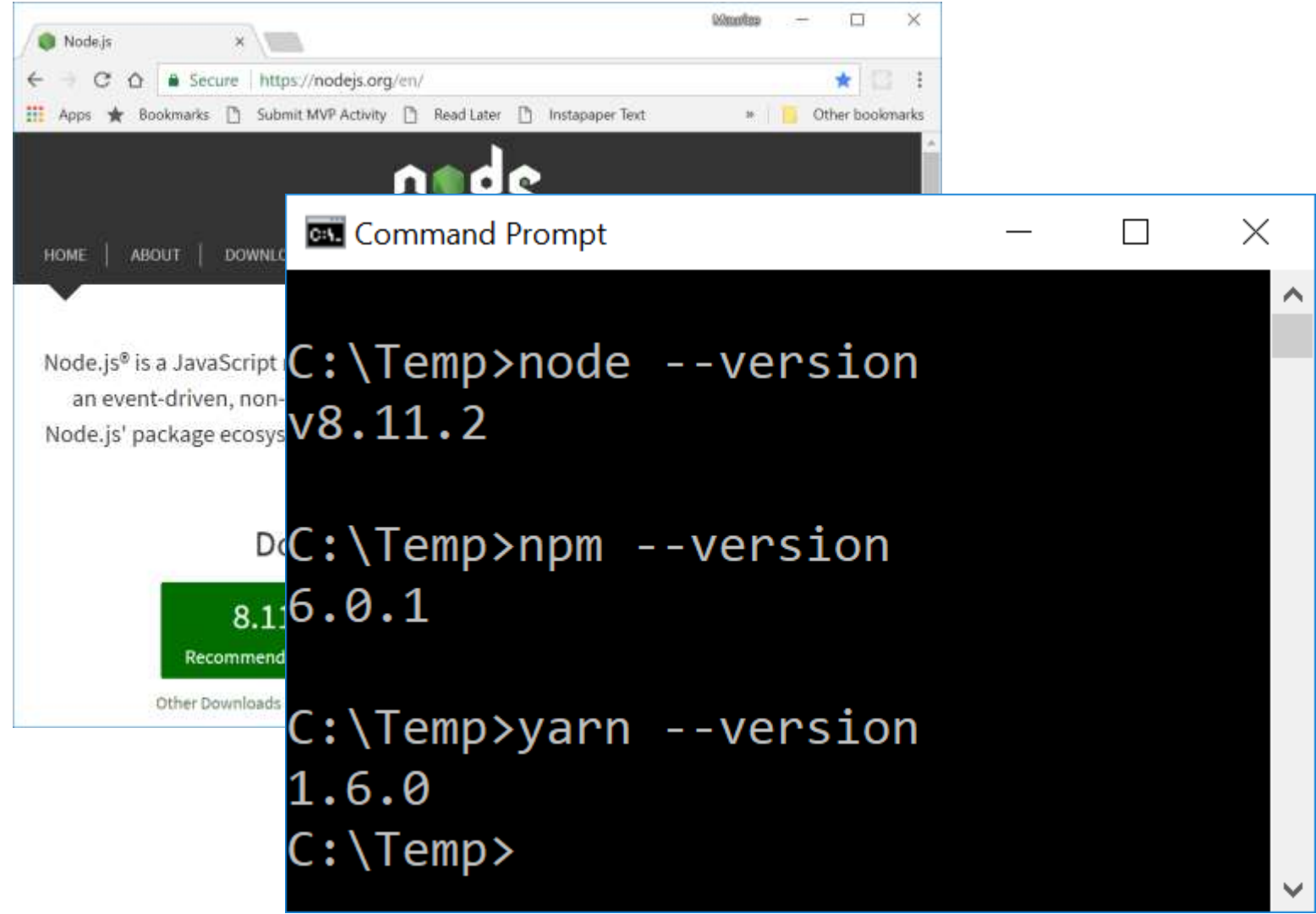
Type it out by hand?

"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

Prerequisites

Install Node & NPM

Install the GitHub repository

Install Node.js & NPM
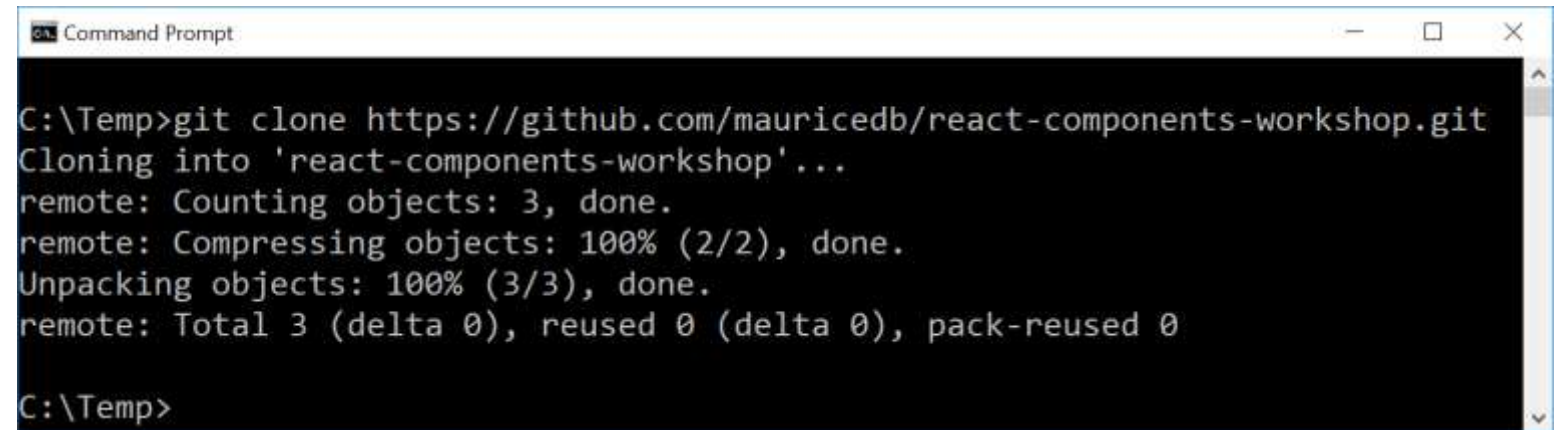
Clone the GitHub repository
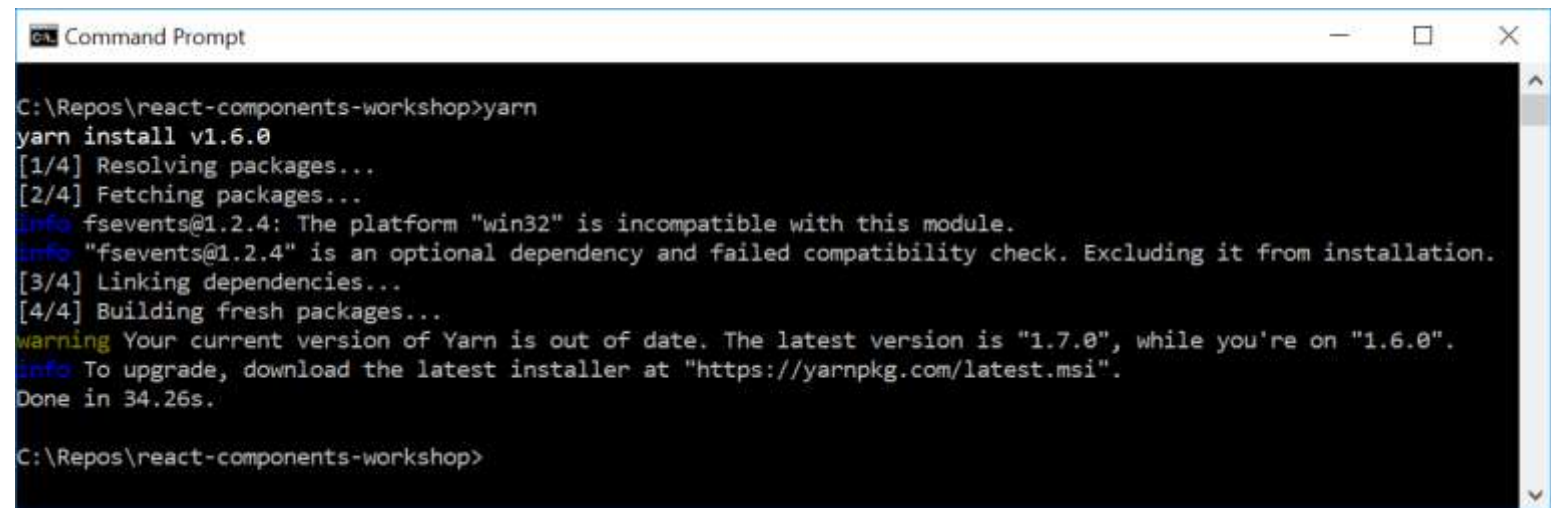
```
Command Prompt                                          —    □    ×

C:\Temp>git clone https://github.com/mauricedb/react-components-workshop.git
Cloning into 'react-components-workshop'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
Unpacking objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

C:\Temp>
```

Install NPM packages

Following along

- Repo: http://bit.ly/rcw-repo
- Slides: http://bit.ly/rcw-slides

Why write better component?

Why?

"A React application is a component tree"

Best practices

- Keep components **small and focused**

- **One component** per file

- Use [Prettier](#) for **formatting**

- Use [StoryBook](#) to UI test components in **isolation**

- Use [Functional Component](#) whenever possible

- Add a **displayName** to each component

Best practices

- Use **ECMAScript 2018** syntax
  - Fat arrow
  - Class properties
  - Object destructuring
  - Spread operator

- Use **type checking**
  - Use propTypes and defaultProps
  - TypeScript or Flow

- Add **unit tests**

Controlled/Uncontrolled components

Uncontrolled input

```jsx
1  import React from 'react';
2
3  const LabeledInput = ({ value, name, label, errorMessage, ...props }) => (
4    <div className="mdl-textfield mdl-js-textfield mdl-textfield--floating-label">
5      <input
6        className="mdl-textfield__input"
7        type="text"
8        id={name}
9        name={name}
10       defaultValue={value}
11       {...props}
12     />
13     <label className="mdl-textfield__label" htmlFor={name}>
14       {label}
15     </label>
16     {errorMessage && (
17       <span className="mdl-textfield__error">{errorMessage}</span>
18     )}
19   </div>
20 );
```

Controlled input

```jsx
import React from 'react';

const LabeledInput = ({
  value,
  name,
  label,
  errorMessage,
  onChange,
  ...props
}) => (
  <div className="mdl-textfield mdl-js-textfield mdl-textfield--floating-label">
    <input
      className="mdl-textfield__input"
      type="text"
      id={name}
      name={name}
      value={value}
      onChange={e => onChange(name, e.target.value)}
      {...props}
    />
    <label className="mdl-textfield__label" htmlFor={name}>
      {label}
    </label>
    {errorMessage && (
      <span className="mdl-textfield__error">{errorMessage}</span>
    )}
  </div>
);
```

Uncontrolled form

```
17    onFormSubmit = e => {
18      e.preventDefault();
19
20      const newMovie = {
21        title: e.target.title.value,
22        director: e.target.director.value,
23        overview: e.target.overview.value
24      };
25
26      alert(JSON.stringify(newMovie, null, 2));
27    };
28
29    render() {
30      const { movie, valid } = this.state;
31
32      return (
33        <div>
34          <h2>Controlled & Uncontrolled components</h2>
35          <form onSubmit={this.onFormSubmit} noValidate>
36            <LabeledInput
37              value={movie.title}
38              name="title"
39              label="Title"
40              required
41              errorMessage="Please enter a title!"
42            />
```

Controlled form

```
17  updateMovie = (name, value) => {
18    const movie = { ...this.state.movie, [name]: value };
19    const valid = !!movie.title && !!movie.director;
20
21    this.setState({ movie, valid });
22  };
23
24  onFormSubmit = e => {
25    e.preventDefault();
26    alert(JSON.stringify(this.state.movie, null, 2));
27  };
28
29  render() {
30    const { movie, valid } = this.state;
31
32    return (
33      <div>
34        <h2>Controlled & Uncontrolled components</h2>
35        <form onSubmit={this.onFormSubmit} noValidate>
36          <LabeledInput
37            value={movie.title}
38            name="title"
39            Label="Title"
40            required
41            errorMessage="Please enter a title!"
42            onChange={this.updateMovie}
43          />
```

Tip

- **Both can be appropriate** in different circumstances
- **Combine** both approaches:
  - How "Controllable" React components maximize reusability
    eFromProps()

Fragments

New in React 16.0

```jsx
 1    import React, { Fragment } from 'react';
 2
 3    const Content = () => (
 4      <Fragment>
 5        <p className="content">
 6          Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum urna
 7          tortor, mollis et turpis ac, convallis imperdiet nulla.
 8        </p>
 9
10
11        <p className="content">
12          In elementum elementum velit blandit luctus. Curabitur vehicula
13          pellentesque odio. Integer gravida imperdiet odio, eget commodo ipsum
14          rutrum in.
15        </p>
16      </Fragment>
17    );
```

Tip

- Use the shorthand **<>** syntax in TypeScript
  - When you don't need to specify any attributes
  - Will be supported by Babel in the future

Error Handling

New in React 16.0

Defining an error boundary

```
You, 3 days ago | 1 author (You)
5   class ErrorBoundary extends Component {
6     static displayName = 'ErrorBoundary';
7
8     state = {
9       error: null,
10      errorInfo: null
11    };
12
13    componentDidCatch(error, errorInfo) {
14      this.setState({ error, errorInfo });
15    }
16
17    render() {
18      const { children } = this.props;
19      const { error, errorInfo } = this.state;
20
21      if (error) {
22        return <ErrorCard error={error} errorInfo={errorInfo} />;
23      }
24
25      return children;
26    }
27  }
```

Using an error boundary

```
6   class ErrorsDemo extends Component {
7       static displayName = 'ErrorsDemo';
8
9       render() {
10          return (
11              <div>
12                  <h2>Error Handling</h2>
13                  <ErrorBoundary>
14                      <ErrorsForm />
15                  </ErrorBoundary>
16              </div>
17          );
18      }
19  }
```

Tip

- The componentDidCatch() is **only for React lifecycle functions** in child components
  - Errors in event handlers, async code etc. are **not caught**

on a server and periodically check

Higher Order Components

Defining a HOC

```
1    import React, { Component } from 'react';
2
3    const withTimeHOC = TheComponent => {
4      return class extends Component {
5        state = { now: new Date() };
6
7        componentDidMount() {
8          this.handle = setInterval(() => this.setState({ now: new Date() }), 1000);
9        }
10
11       componentWillUnmount() {
12         clearInterval(this.handle);
13       }
14
15       render() {
16         const { now } = this.state;
17         const { now } = this.state;
18
19         return <TheComponent now={now} />;
20       }
21     };
22   };
```

Using a HOC

```
3    import SimpleClock from '../../components/SimpleClock';
4    import AnalogClock from '../../components/AnalogClock';
5    import withTimeHOC from './withTimeHOC';
6
          rentTime = withTimeHOC(SimpleClock);
          rentTime = withTimeHOC(AnalogClock);
```

Render Props

Defining a Render Prop

```
1   import React, { Component } from 'react';
2
    You, 4 days ago | 1 author (You)
3   class WithTimeRP extends Component {
4     state = { now: new Date() };
5
6     componentDidMount() {
7       this.handle = setInterval(() => this.setState({ now: new Date() }), 1000);
8     }
9
10    componentWillUnmount() {
11      clearInterval(this.handle);
12    }
13    render() {
14      const { children } = this.props;
15      const { now } = this.state;
16
17      return children(now);
18    }
19  }
```

Using a Render Prop

```
48    <WithTimeRP>
49      {now => (
50        <div className="mdl-cell mdl-cell--4-col">
51          <SimpleClock now={now} />}
52          <AnalogClock time={now} />
53        </div>
54      )}
55    </WithTimeRP>
```

Tip

- Higher Order Functions are **easy to use**
  - Complexity is for developer of HOC

- Render Props are **more flexible**
  - ...ming developer

New Context API

New in React 16.3

Creating the context

```
1   import React, { Component, createContext } from 'react';
2
3   const TimeContext = createContext();
4
5   export const TimeContextConsumer = TimeContext.Consumer;
6
7   export class TimeContextProvider extends Component {
8     state = { now: new Date() };
9
10    componentDidMount() {
11      this.handle = setInterval(() => this.setState({ now: new Date() }), 1000);
12    }
13
14    componentWillUnmount() {
15      clearInterval(this.handle);
16    }
17    render() {
18      const { children } = this.props;
19      const { now } = this.state;
20
21      return <TimeContext.Provider value={now}>{children}</TimeContext.Provider>;
22    }
23  }
```

Using the context

```
6   import { TimeContextProvider, TimeContextConsumer } from './TimeContext';
7
8   class ErrorsDemo extends Component {
9     render() {
10      return (
11        <div>
12          <TimeContextProvider>
13            <h2>Context API</h2>
14            <div className="mdl-grid">
15              <div className="mdl-cell mdl-cell--2-col" />
16              <div className="mdl-cell mdl-cell--4-col">
17                <TimeContextConsumer>{now => <SimpleClock now={now} />}</TimeContextConsumer>
18              </div>
19              <div className="mdl-cell mdl-cell--4-col">
20                <TimeContextConsumer>{now => <AnalogClock time={now} />}</TimeContextConsumer>
21              </div>
22            </div>
23          </TimeContextProvider>
24        </div>
25      );
26    }
27  }
```

🔥 Tip

- The context can contain **both data and functions**
- Make sure you don't create a new context object every render

React Profiler

New in React 16.4

Experimental!

Profiler component

```jsx
1   import React, { Component, unstable_Profiler as Profiler } from 'react';
2
3   class MyProfiler extends Component {
4     static defaultProps = {
5       id: 'profile'
6     };
7
8     onRender = (id, phase, actualTime) => console.log(id, phase, actualTime);
9
10    render() {
11      const { children, id } = this.props;
12
13      return (
14        <Profiler id={id} onRender={this.onRender}>
15          {children}
16        </Profiler>
17      );
18    }
19  }
```

Profiling child components

```jsx
17    render() {
18      const { firstName, primeCount } = this.state;
19
20      return (
21        <MyProfiler>
22          <div>
23            <h2>Profiler</h2>
24
25            <div>
26              <MyProfiler id="inputs">
27                <LabeledInput
28                  label="Max Prime:"
29                  value={primeCount}
30                  type="number"
31                  name="primeCount"
32                  onChange={this.onChange}
33                />
34              </MyProfiler>
35              <MyProfiler id="primes">
36                <Primes primeCount={primeCount} />
37              </MyProfiler>
38            </div>
39          </div>
40        </MyProfiler>
41      );
42    }
43  }
```

⚡ Tip

- Provides timing information on **each update**
  - As well as the initial mount

- Profiler components can be **nested** as required
  - ll child components

Pure Component

Benefits

- Normal components **render every time the parent renders**
  - Even if props have not changed

- A PureComponent **only renders when the input props change**
  - Or it's own state is updated

```jsx
import React, { PureComponent } from 'react';

class Primes extends PureComponent {
  calculatePrimes() {
    const primes = [];
    // Calculating primes
    return primes;
  }
  render() {
    const { primeCount } = this.props;
    const primes = this.calculatePrimes();

    return (
      <div>
        <h3>Primes</h3>
        {primes.join(', ')}
      </div>
    );
  }
}

export default Primes;
```

🔥 Tip

- **All props are checked**, including children using ===
  - JSX children are always new objects
 - Inline fat arrow are also always new objects

getDerivedStateFromProps

New in React 16.3

```
4   class Primes extends Component {
5     state = { primeCount: 0, primes: [] };
6
7     static getDerivedStateFromProps(props, state) {
8       if (props.primeCount !== state.primeCount) {
9         const { primeCount } = props;
10        const primes = calculatePrimes(primeCount);
11
12        return { primeCount, primes };
13      }
14
15      return null;
16    }
17
18    render() {
19      const { primes } = this.state;
20
21      return (
22        <div>
23          <h3>Primes</h3>
24          {primes.join(', ')}
25        </div>
26      );
27    }
28  }
```

⌂ Tip

- A very flexible way to prevent recalculation of values
  - Often a better approach then PureComponent
  - But can be combined if required
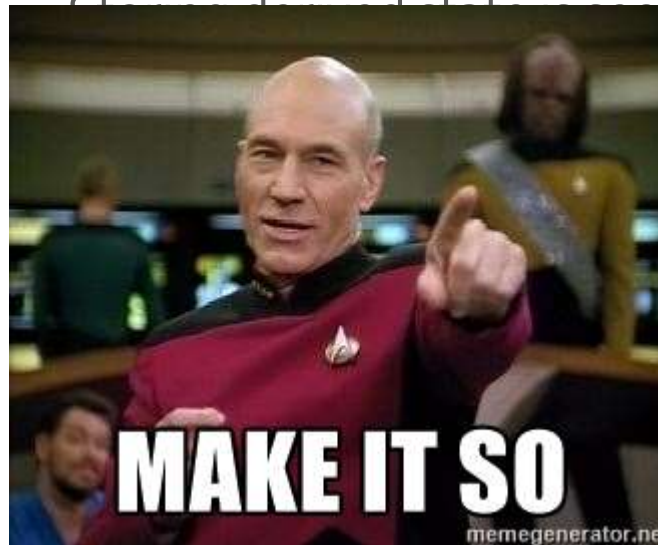
- Storing derived state is ~~seen~~ as an anti-pattern
  - ~~ded~~

Maurice de Beijer

@mauricedb

maurice.de.beijer
@gmail.com