

# 03 - Components

DEV4 - 2019-2020

# Components

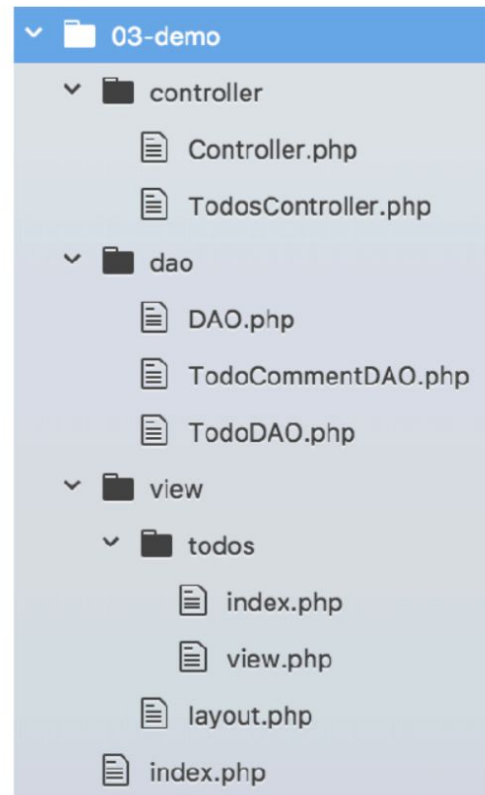
*Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.*

- Instead of artificially separating **technologies** by putting markup and logic in separate files, React separates **concerns** with loosely coupled units called “**components**” that contain **both**.

# Separating technologies by putting markup and logic in separate files

## → MVC

- ◆ Model                      Data, database
- ◆ View                        Hoe data weergeven, templates
- ◆ Controller                Interactie



# Separation of concern

*React separates concerns with loosely coupled units called “components” that contain **both**.*

- Markup in JS? Alles samen in 1 file?
  - ◆ JSX



*Ok, en wat is een*

# **Component**

*nu precies?  
in code...*

# Component

*Components are like JavaScript functions.  
They accept arbitrary inputs (called “props”) and return  
React elements describing what should appear on the screen.*

→ Een functie die een element terug geeft is al een component

```
const Hello = properties => <h1>Hello {properties.planet}!</h1>;
```

# Component

→ Je kan een component ook schrijven als een ES6 class

```
class Hello extends React.Component {  
  render() {  
    return <h1>Hello {this.properties.planet}!</h1>;  
  }  
}
```

That's so 2018!

# Hoe kunnen we een component renderen?

- React elementen bestaan niet enkel uit DOM tags
- Een React element kan ook een component zijn

```
const Hello = props => <h1>Hello {props.planet}!</h1>;
```

```
const element = <Hello planet="World"/>;
```



# Hoe kunnen we een component renderen?

```
import React from 'react';
import {render} from 'react-dom';

const Hello = props => <h1>Hello {props.planet}!</h1>;

const element = <Hello planet="World"/>;

render(
  element,
  document.querySelector(`#root`)
);
```

# Hoe kunnen we een component renderen?

→ JSX attributen worden aan een component doorgegeven als een object, 'props' genaamd

```
import React from 'react';
import {render} from 'react-dom';

const Hello = props => <h1>Hello {props.planet}!</h1>;

const element = <Hello planet="World"/>;

render(
  element,
  document.querySelector(`#root`)
);
```

A diagram consisting of two blue curved arrows. The first arrow starts from the 'props' parameter in the function definition 'const Hello = props =>' and points to the 'props' object in the JSX element '<Hello {props.planet}!</h1>'. The second arrow starts from the 'planet' prop in the JSX element '<Hello planet="World"/>' and points to the 'props.planet' property access in the function body '{props.planet}'.

<https://codesandbox.io/s/31j0mp7om5>



# Componenten combineren

# Componenten combineren

```
const Hello = props => <h1>Hello {props.planet}!</h1>;
```

```
const App = () => {  
  return (  
    <div>  
      <Hello planet="Venus"/>  
      <Hello planet="Earth"/>  
      <Hello planet="Mars"/>  
    </div>  
  )  
}
```

```
ReactDOM.render(  
  <App />,  
  document.querySelector(`#root`)  
);
```

<https://codesandbox.io/s/oxmjprk77z>

# Component


- Starten met een hoofdletter
- 1 root element (kan ook <> zijn)
- Niet bang zijn om een component te maken
  - ◆ Stukje UI dat op verschillende plaatsen gebruikt is? => Component
  - ◆ Complex genoeg op zijn eigen? => Component



**Props**

# Props

- Properties die een component binnen krijgt
- Een component mag nooit zijn props aanpassen
- Componenten moeten altijd pure functions zijn
  - ◆ Nooit props aanpassen
  - ◆ Bij dezelfde props altijd dezelfde output genereren.



```
hello = props => <h1>Hello {props.planet}!</h1>  
component = <Hello planet="World"/>;
```

The diagram illustrates the flow of props in a functional component. A light blue curved arrow originates from the `props` argument in the function call `component = <Hello planet="World"/>;` and points to the `props` parameter in the function definition `hello = props => <h1>Hello {props.planet}!</h1>`. A second light blue curved arrow originates from the `props.planet` expression within the JSX element in the function definition and points back to the `props` parameter, demonstrating how the prop is accessed within the function.

# Propvol props

- Alle attributen zitten samen in 1 'props' object
- Destructuring FTW!

```
<Student  
  naam="Jantje"  
  richting="Devine"  
  groep="2DEV5"  
>
```

```
const Student = props => {  
  const naam = props.naam;  
  const richting = props.richting;  
  const groep = props.groep;  
  return <p>`${naam} zit in ${groep} van ${richting}`</p>;  
};
```

```
const Student = ({ naam, richting, groep }) => (  
  <p>`${naam} zit in ${groep} van ${richting}`</p>  
);
```





- Alle overige props terug in een 'props' object stoppen (rest)

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest\\_parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters)

```
const MyComponent = ({ specificProp, ...props }) => ( more code here
```

- Alle props doorgeven aan een child component (spread)

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax)

```
<MyOtherComponent {...props} />
```



# Functions in props

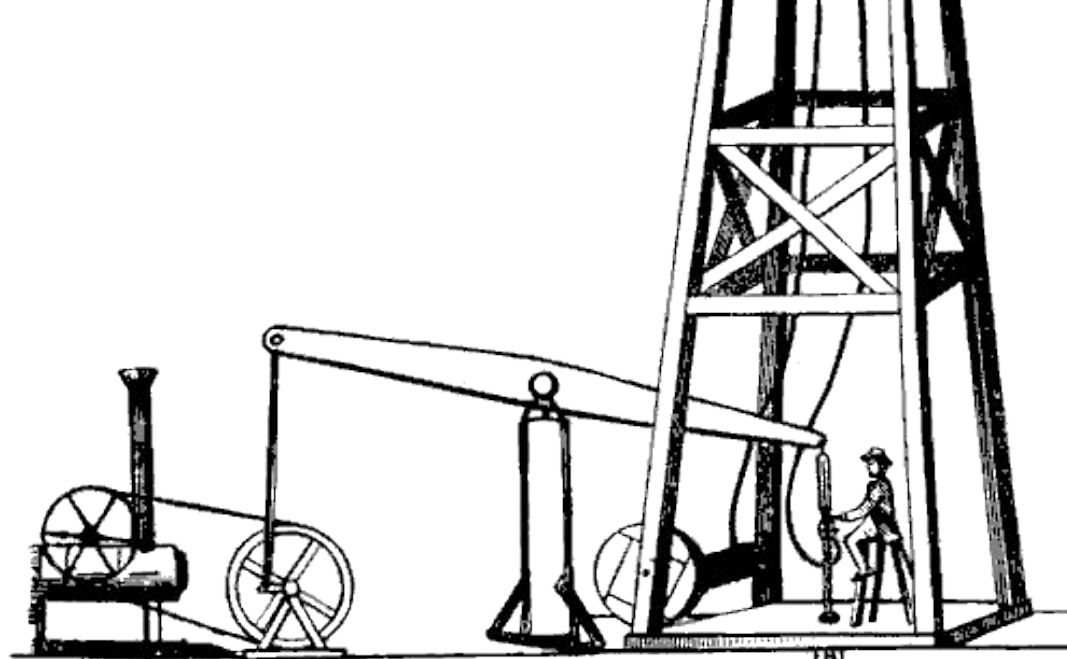
# Functies in props

- Een referentie naar een functie is ook een variabele
- ◆ Dus je kan een functie ook doorgeven naar een child component.
  - ◆ Zo kan een child component een functie aanroepen in een parent component.

```
const callMe = name => {  
  ...  
};  
return <Child onCall={callMe} />
```

```
const Child = ({ onCall }) => {  
  return <button onClick={() => onCall("Me")}>  
    Click Me  
  </button>;  
};
```

# Prop Drilling



component

> component

> component

> component

> component



# Typechecking

# Typechecking

- JavaScript is a loosely typed or a dynamic language
  - ◆ Handig maar ook gevaarlijk
- Een component die een Number verwacht maar een String krijgt.
  - ◆ Bugs!
- Controleren van welk type een variabele is
  - ◆ Typechecking
- Controleren van welk type een property in de props is
  - ◆ PropTypes

# PropTypes

→ `import PropTypes from "prop-types";`

→ checken met PropTypes.

- ◆ array
- ◆ bool
- ◆ func
- ◆ number
- ◆ object
- ◆ string
- ◆ symbol
- ◆ <https://reactjs.org/docs/typechecking-with-proptypes.html>

→ chainen met `.isRequired`

- ◆ `PropTypes.string.isRequired`

# PropTypes in action

```
import React from "react";
import PropTypes from "prop-types";

const Som = ( { a, b } ) => {
  return ( <p>{a} + {b} = {a + b}</p> );
}

Som.propTypes = {
  a: PropTypes.number.isRequired,
  b: PropTypes.number.isRequired
};

export default Som;
```



# PropTypes fouten

→ Worden getoond in console (!)

→ `<Som b={2}/>`

```
✖ Warning: Failed prop type: The prop `a` is marked as required in `Som`, but its value is `undefined`.
  in Som (at App.js:8)
  in App (at index.js:7)
```

→ `<Som a="1" b={2}/>`

```
✖ Warning: Failed prop type: Invalid prop `a` of type `string` supplied to `Som`, expected `number`.
  in Som (at App.js:8)
  in App (at index.js:7)
```

# Default Props

```
import React from "react";
import PropTypes from "prop-types";

const Som = ( { a, b } ) => {
  return ( <p>{a} + {b} = {a + b}</p> );
}
```

```
Som.defaultProps = { a: 1, b: 1};
```

```
Som.propTypes = {
  a: PropTypes.number.isRequired,
  b: PropTypes.number.isRequired
};
```

```
export default Som;
```

<Som b={2}/>

1 + 2 = 3

<https://codesandbox.io/s/j4lr4pjq9>




**State**

# State

- Interne status van een component
  - ◆ Data om het component mee op te bouwen
- Informatie/data voor een component die kan veranderen
  - ◆ Zal typisch veranderen door een user/system event (user input, server response,...)
- Was vroeger (<2018) enkel beschikbaar in *Class components*
- 2 delen nodig
  - ◆ De waarde bijhouden => variabele
  - ◆ De waarde aanpassen => functie

# useState



REACT  
CONF 2018

Name  
Mary

```
import React from 'react';
import Row from './Row';

export default function Greeting(props) {
  const [name, setName] = ???

  function handleNameChange(e) {
    setName(e.target.value);
  }

  return (
    <section>
      <Row label="Name">
        <input
          value={name}
          onChange={handleNameChange}
        />
      </Row>
    </section>
  );
}
```

<https://www.youtube.com/watch?v=dpw9EHDh2bM&feature=youtu.be&t=1221>

(20:21 - 22:35)

# useState

- `useState` is een 'hook' (beschikbaar vanaf 2018)
- `import React, { useState } from "react";`
- `useState(startwaarde)`
  - ◆ Geeft een array terug met op index:
    - 0: de waarde
    - 1: de functie om die waarde te overschrijven
- Dus, via array-destructuring:
  - ◆ `const [waarde, functieOmWaardeAanTePassen] = useState(startwaarde);`
  - ◆ `const [name, setName] = useState("Jantje");`



## 2 regels voor 'hooks'

### Only Call Hooks at the Top Level

**Don't call Hooks inside loops, conditions, or nested functions.** Instead, always use Hooks at the top level of your React function. By following this rule, you ensure that Hooks are called in the same order each time a component renders. That's what allows React to correctly preserve the state of Hooks between multiple `useState` and `useEffect` calls. (If you're curious, we'll explain this in depth [below](#).)

### Only Call Hooks from React Functions

**Don't call Hooks from regular JavaScript functions.** Instead, you can:

-  Call Hooks from React function components.
-  Call Hooks from custom Hooks (we'll learn about them [on the next page](#)).

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

# State

- Met useState of met MobX?
- 'overal' nodig?
  - ◆ MobX
- Eenmalig nodig (1 component?)
  - ◆ useState
  - ◆ bv: voor controlled components van een form



# Tijd voor een verhaaltje in React

*Zonder MobX*

# Neem nu de volgende app:

`<App/>`

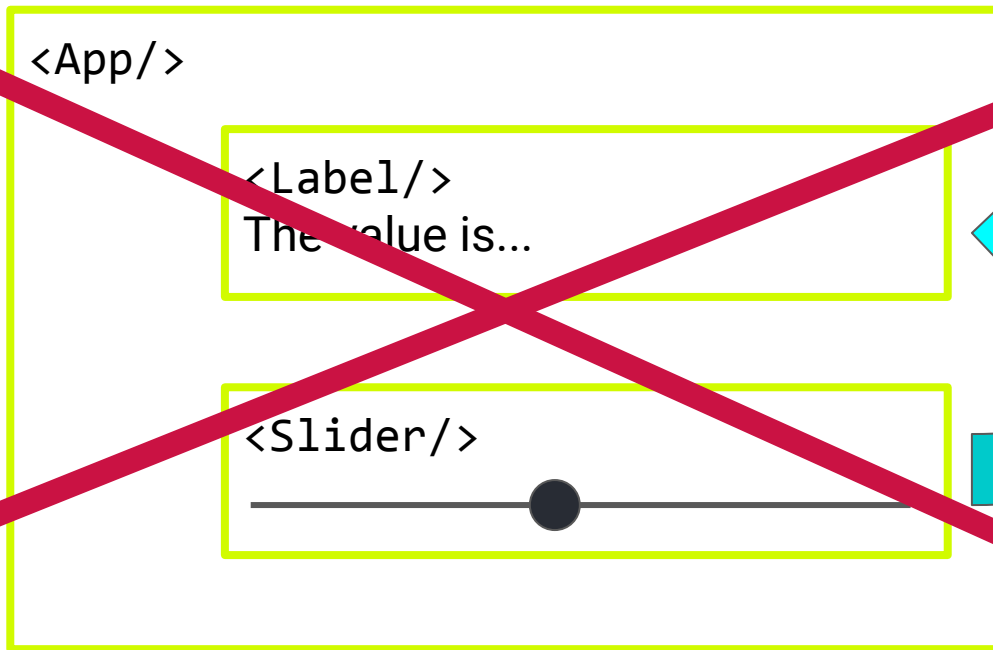
`<Label/>`

The value is...

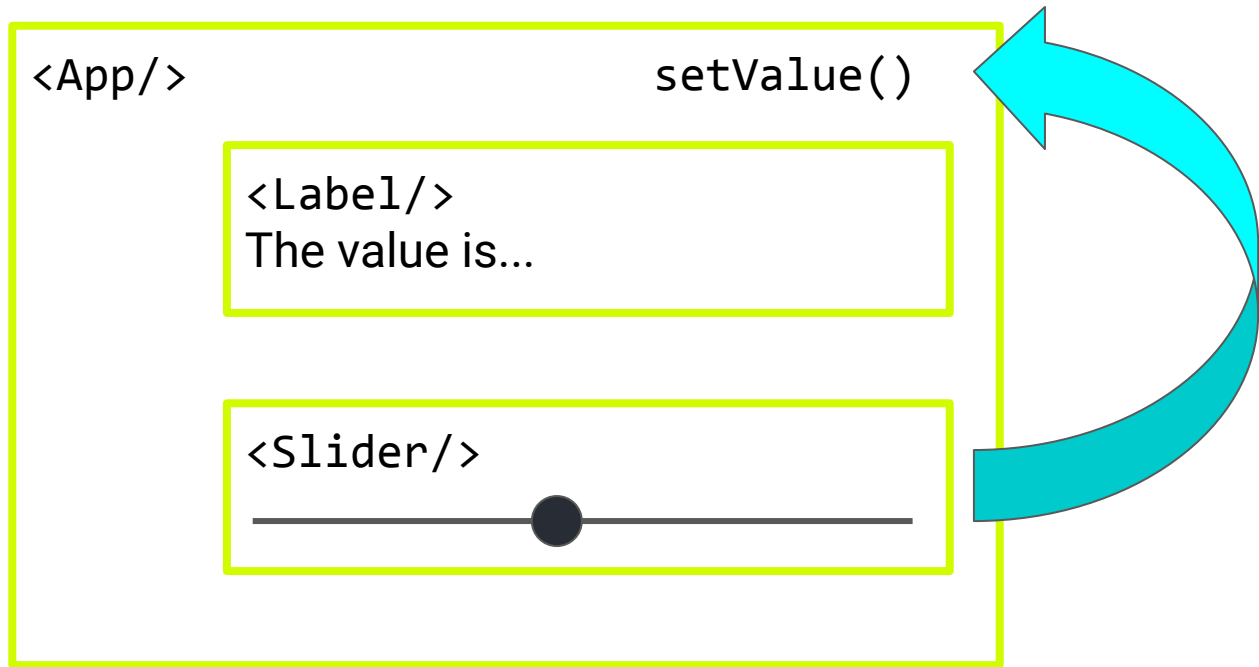
`<Slider/>`



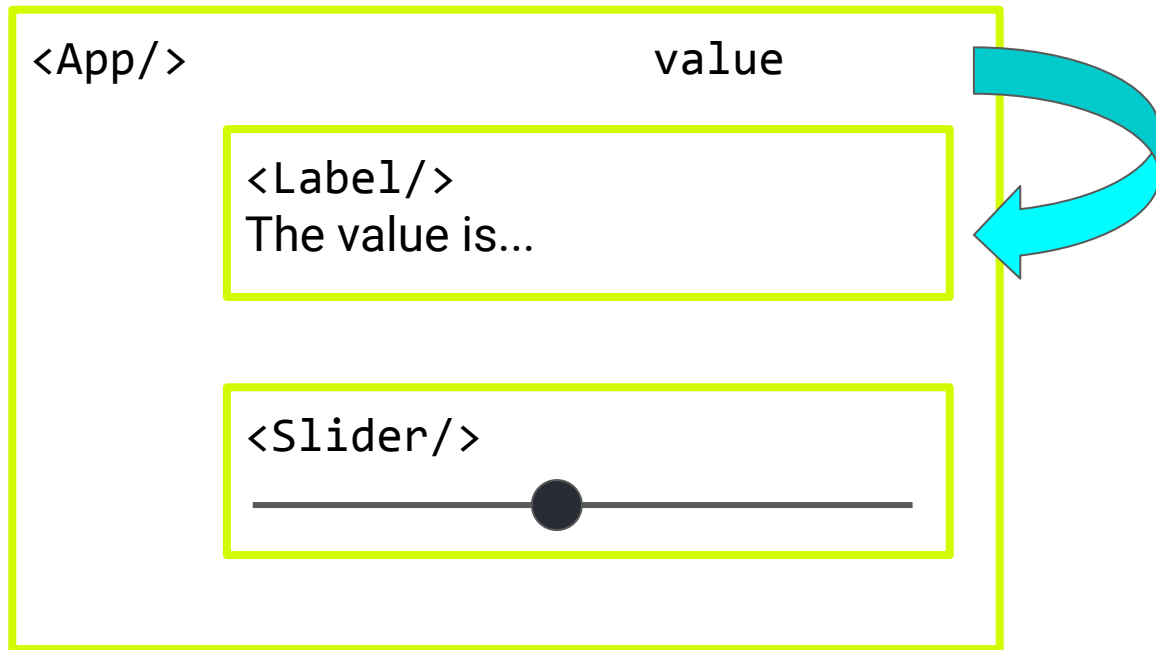
# De gebruiker past de slider aan, label moet veranderen



# Communicatie via container element



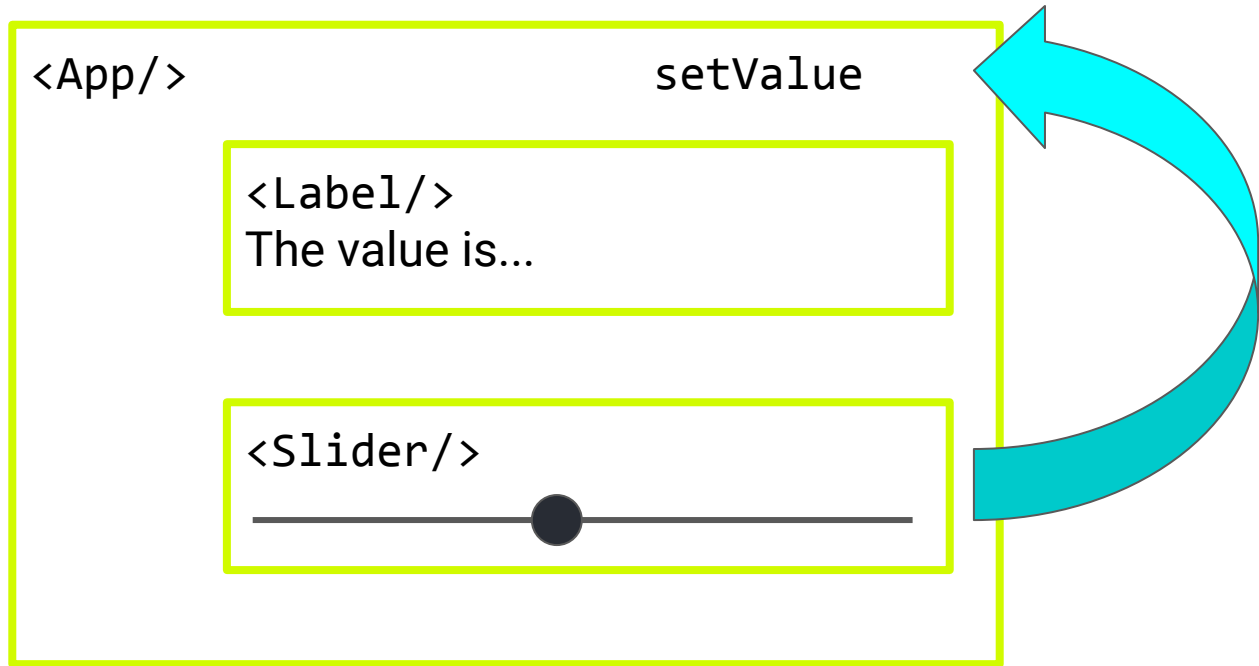
# State doorgeven als prop





**Ok, break it down**

# 1: state in <App/> manipuleren



<App>

<Label/>

<Slider/>

</App>





# Child => parent communicatie

- Events van child element opvangen in parent element
  - ◆ Handlers
- Props doorgeven aan de state
  - ◆ props => state
  - ◆ handler op `<Slider/>`    state in `<App/>`

prop op <Slider/>



functie in <App/>

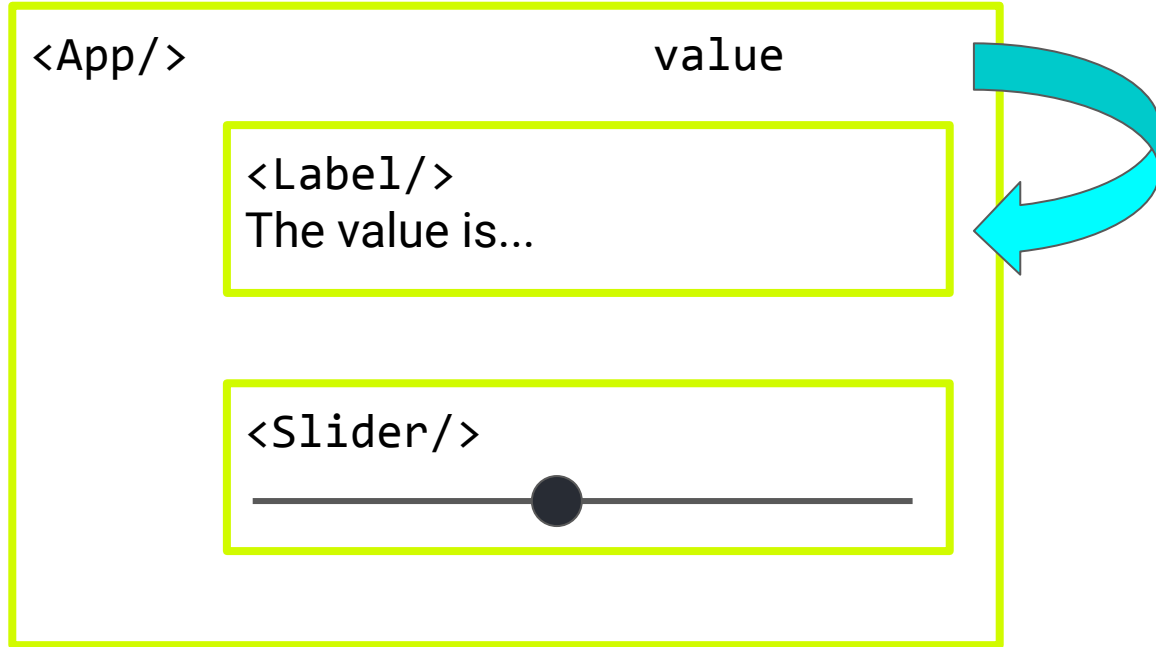


```
<Slider onChangeSlider={setValue} />
```



state in <App/> aangepast

## 2: State doorgeven als prop



<App>

<Label/>

<Slider/>

</App>



# Parent => child communicatie

- Aanpassen van de state (in stap 1)
  - ◆ render wordt getriggerd
- Data doorgeven aan componenten
  - ◆ state => props
  - ◆ State uit `<App />`      prop op `<Label/>`

prop op <Label/>



state uit <App/>



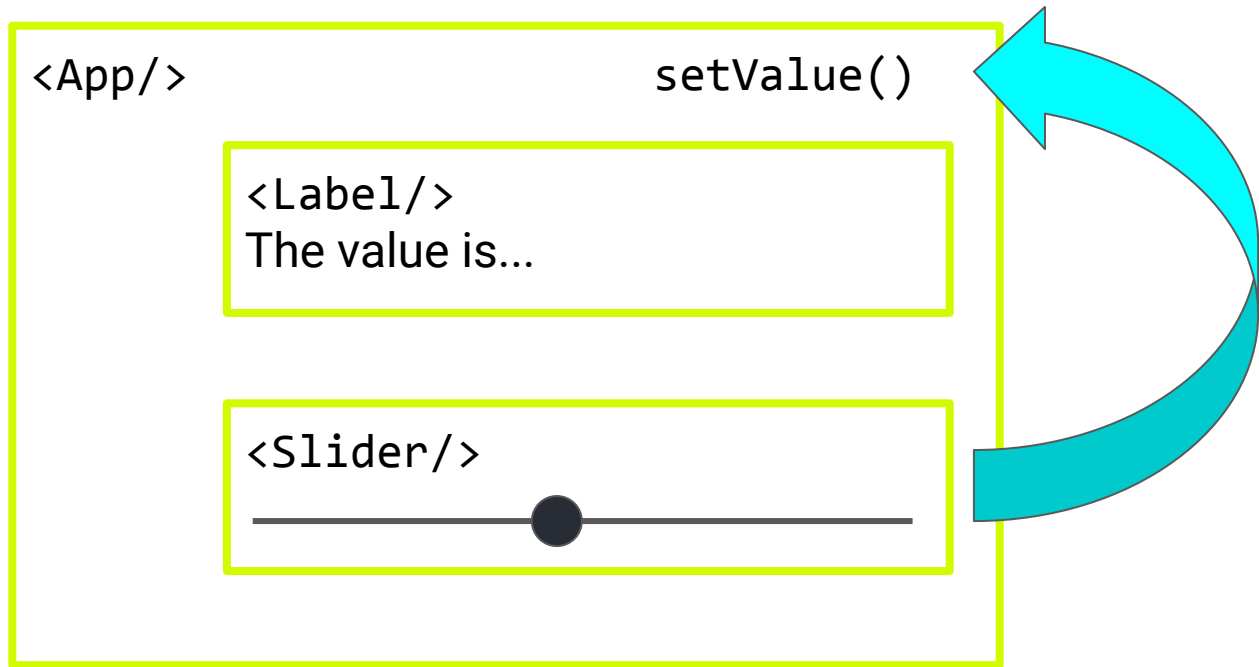
<Label value={value} />

# Unidirectional dataflow



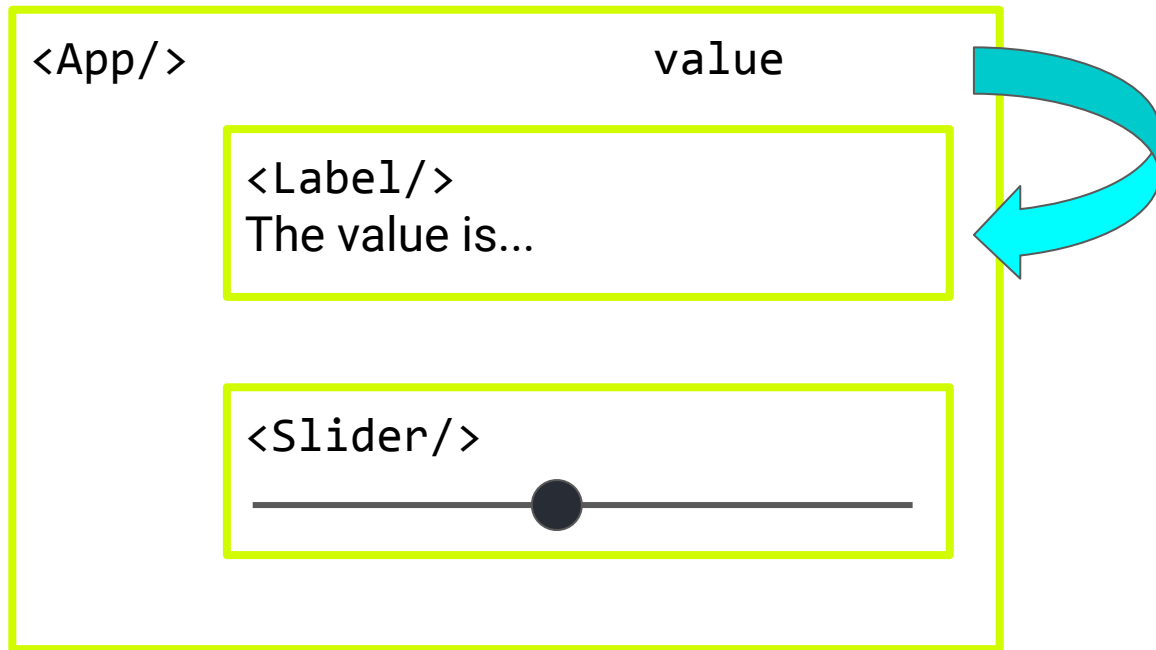
<https://codesandbox.io/s/j1v13o0jwv>

# Communicatie via container element





# State doorgeven als prop



# 'domme' componenten

- Gebruiken props om UI te renderen
- Geen / weinig logica

<App>

<Label/> ←

<Slider/> ←

</App>

# 'slimme' componenten

- Gebruiken props + state om UI te renderen
- logica

<App> ←

<Label/>

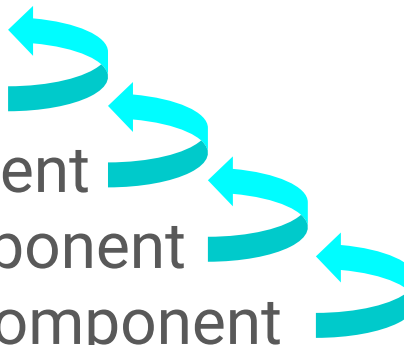
<Slider/>

</App>

# More state, more problems

- State overal in elk component bijhouden == héél slecht idee
  - ◆ Moeilijk om in sync te houden over componenten heen
- Zo veel mogelijk 'domme' componenten gebruiken

component  
 > component  
 > component  
 > component  
 > component

A diagram illustrating the return flow of nested function calls. It shows a list of five 'component' calls, each indented further to the right than the previous one. To the right of each call is a thick, curved blue arrow pointing back to the left, towards the previous level of indentation. This visualizes how each nested call returns control to its caller.

# Unidirectional dataflow

handler doorgeven aan parent die  
handler doorgEEft aan parent die  
handler doorgEEft aan parent die  
handler doorgEEft aan parent die  
handler doorgEEft aan parent die  
...

# Unidirectional dataflow

→ *Lifting state up, the data flows down*

- ◆ <https://reactjs.org/docs/state-and-lifecycle.html#the-data-flows-down>
- ◆ <https://reactjs.org/docs/lifting-state-up.html>

→ Kan veel typewerk worden

- ◆ Veel doorgeven, herhalen
- ◆ Er bestaan hier oplossingen voor...



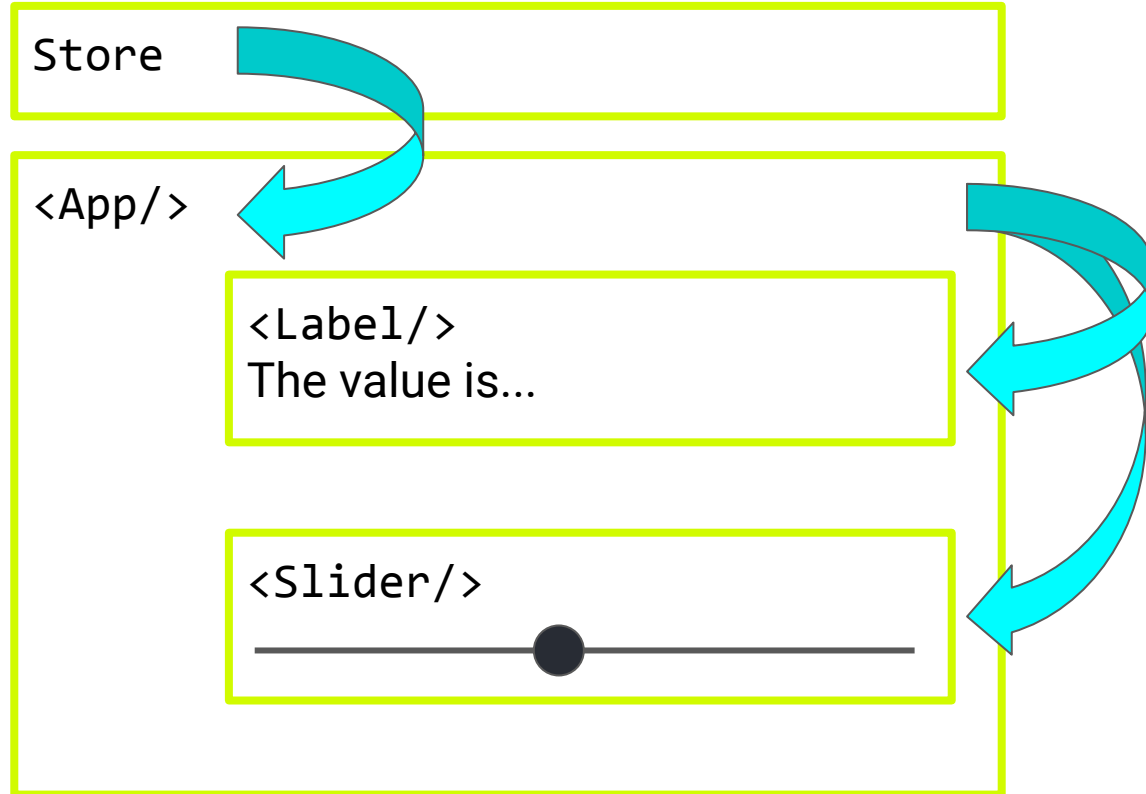
**En euh, wat met MobX?**



# Tijd voor een verhaaltje in React

*Mét MobX*

# Pass the store



# Pass the store

→ *the data (store) flows down*

- ◆ Via de actions in de store kan je de state aanpassen

→ De volledige store doorgeven

- ◆ <https://codesandbox.io/s/data-flow-mobx-store-3zw5o>

→ Properties van de store doorgeven

- ◆ <https://codesandbox.io/s/data-flow-mobx-parts-vkuxi>



# UI Store

# Verschillende soorten data/state

## → Message

- ◆ content
- ◆ user
- ◆ isRead

## → Post

- ◆ title
- ◆ description

## → currentMessage

## → activeFilter

## → loggedInUser

## → darkMode

## → language

(Domain) Data

User interface

# Aparte store

- Alles in één store plaatsen is niet (meer) overzichtelijk
- Verschillende stores maken en samenvoegen

- UserStore, MessageStore, GroupStore => DataStore?
- UiStore

- 'RootStore' bevat de verschillende stores
  - ◆ Maar 1 instantie van maken

Demo: <https://codesandbox.io/s/data-flow-mobx-uistore-tl6pu>  
<https://mobx.js.org/best/store.html> (zelfde principe, next level uitvoering)



**create-react-app**

# Create react app

<https://github.com/facebook/create-react-app>

```
npx create-react-app my-app
```

```
cd my-app
```

```
yarn start
```

of...

spatie



```
yarn create react-app my-app
```

```
cd my-app
```

```
yarn start
```

```
λ npx create-react-app my-app
npx: installed 114 in 4.308s

Creating a new React app in ~/my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...

yarn add v1.2.1
info No lockfile found.
[1/4] 🔍 Resolving packages...
[2/4] 📦 Fetching packages...
[3/4] 🔗 Linking dependencies...
```



# react-scripts

- Verzameling van packages en config
  - ◆ <https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/package.json>
- It just works
- Kan je in één keer updaten
  - ◆ yarn outdated

```
"dependencies": {  
  "@testing-library/jest-dom": "^4.2.4",  
  "@testing-library/react": "^9.3.2",  
  "@testing-library/user-event": "^7.1.2",  
  "react": "^16.12.0",  
  "react-dom": "^16.12.0",  
  "react-scripts": "3.4.0",  
},  
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

# Files...

- index.js
  - ◆ Bevat de `ReactDOM.render(<App />, document.getElementById('root'));`
- App.js
  - ◆ Rootcomponent
- App.css, index.css
  - ◆ Aparte css files
- App.test.js, setupTests.js
  - ◆ Components testen (later meer)
- Logo.svg
  - ◆ `import logo from './logo.svg';`
  - ◆ `<img src={logo} className="App-logo" alt="logo" />`
- ServiceWorker.js
  - ◆ <https://create-react-app.dev/docs/making-a-progressive-web-app/>

> node\_modules

> public

▼ src

# App.css

JS App.js

JS App.test.js

# index.css

JS index.js

🖼 logo.svg

JS serviceWorker.js

JS setupTests.js

📄 .gitignore

{ } package.json

📖 README.md

👤 yarn.lock

# (MobX) Decorators in create-react-app

<https://github.com/facebook/create-react-app/issues/411>



gaearon commented on 26 Aug 2016

Member



Our position is simple: we add transforms that are either stable enough (like `async/await`) or heavily used by Facebook (like class properties). Only that lets us be confident in suggesting them, because if something changes in the standard, we'll write and release a codemod to migrate away from them.

...

So no, we won't provide support for them until they either become a part of the language, or we start using them internally at Facebook. Sorry! Bring it up with the tutorial authors because MobX absolutely does not *require* decorators.



17



3



gaearon commented on 1 Sep 2016 • edited ▼

Member



For people coming to this thread later—if you use MobX or a similar library, you don't **need** decorators. They are just syntax sugar in this case.

Learn more: [facebookincubator#214 \(comment\)](#), [facebookincubator#411 \(comment\)](#).

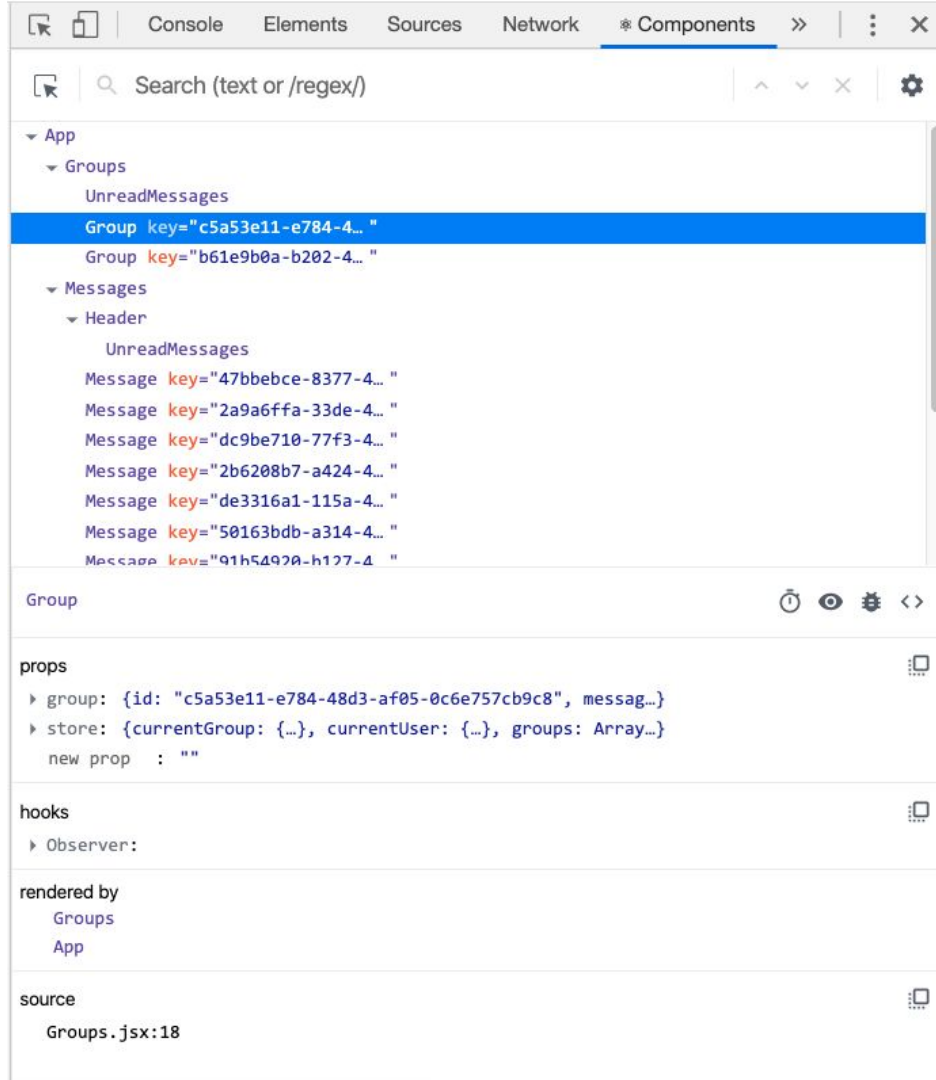
Here is why we don't include them: [facebookincubator#411 \(comment\)](#).



# React Developer Tools

# React Developer Tools

→ Chrome Extensie





**uuid**

# Id nodig?

→ Uniek id nodig voor entities

- ◆ id++?
- ◆ new Date()?
- ◆ hmmm...

→ yarn add uuid

→ <https://www.npmjs.com/package/uuid#ecmascript-modules>



# Documentatie



# Docs

- <https://reactjs.org/docs/getting-started.html>
- <https://create-react-app.dev/docs/getting-started>
- Class Components? (this.setState())
  - ◆ Kan je herschrijven naar functional components
  - ◆ Vervangen door useState()



# Opdracht

# Opdracht Week03

- Splits jouw app op in verschillende components
- PropTypes voorzien
- Store doorgeven via props
- Store opsplitsen in DataStore & UiStore



# Hulp nodig? Tips voor issues

- Een issue heeft een titel + een description...
- Altijd handig als je kan tonen waar het mis gaat
  - ◆ Gebruik permalinks



A screenshot of a code editor with a context menu open over a code block. The code block contains the following JavaScript code:

```
73 const init = () => {  
74   const store = new Store();  
75   autorun(() => {  
76     renderGroups(store);  
77     renderMessages(store.currentMessages);  
78     (store.currentGroup);  
79   });  
80 }  
81  
82 export default {  
83   init,  
84   Group({name: 'vrienden', pic:  
85     [new Message({content: 'Hey', u
```

The context menu has the following options:

- Copy lines
- Copy permalink
- View git blame
- Reference in new issue



A screenshot of a code preview window. The window has two tabs: "Write" and "Preview". The "Preview" tab is selected. The preview shows the following code block:

```
76   renderGroups(store);  
77   renderMessages(store.currentMessages);  
78   renderHeader(store.currentGroup);
```

Below the code block, the following text is displayed:

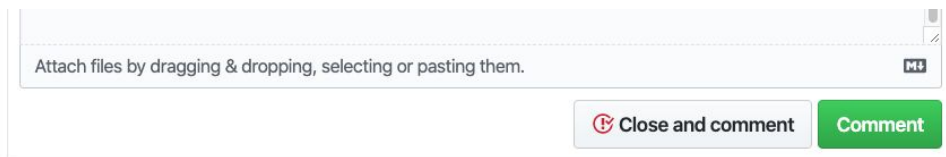
DEV4-2020/02 - Mobx/docent/04 - ThatsApp groups/src/index.js  
Lines 76 to 78 in 79b4aa1

# Hulp nodig? Tips voor issues



→ Labelen met 'help wanted'


→ Als wij antwoorden, dan sluiten we de issue ook.

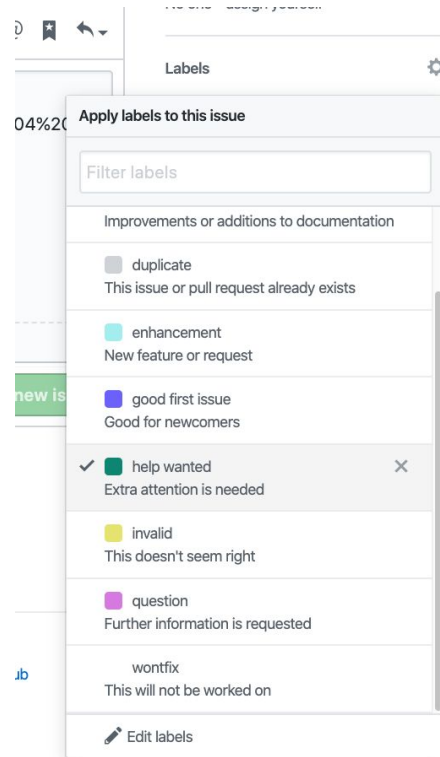
- ◆ Zo behouden wij overzicht
- ◆ Nog niet duidelijk?
  - Terug op replyen, dan is de issue weer open



Attach files by dragging & dropping, selecting or pasting them.

 Close and comment 

 Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).





# Development

# React & MobX

## → Dan Abramov

- ◆ React team
- ◆ [https://twitter.com/dan\\_abramov](https://twitter.com/dan_abramov)
- ◆ [#reactgate](#)



## → Michel Weststrate

- ◆ MobX
- ◆ <https://twitter.com/mweststrate>

