Joren Jamar

AP Valley 2018  3EA1

React cursus



Inhoudsopgave

[React app aanmaken 2](#_Toc527929392)

[App runnen 2](#_Toc527929393)

[Bootstrap in applicatie instalen 2](#_Toc527929394)

[In index.js toevoegen: 2](#_Toc527929395)

[Component toevoegen 2](#_Toc527929396)

[In naamVanComponent.jsx 2](#_Toc527929397)

[Voeg toe aan index.js 3](#_Toc527929398)

[2 elementen meegeven aan component 3](#_Toc527929399)

[Dynamisch tonen van value (in component) 4](#_Toc527929400)

[Voorbeeld met statische tekst 4](#_Toc527929401)

[Voorbeeld met wiskundige formule 4](#_Toc527929402)

[Voorbeeld met oproepen van functie die waarde terug geeft 5](#_Toc527929403)

[Setting Attributes 5](#_Toc527929404)

[Foto dynamisch instellen 5](#_Toc527929405)

[Bootstrap styling gebruiken/class gebruiken 6](#_Toc527929406)

[Styles aanpassen optie 1 6](#_Toc527929407)

[Styles aanpassen optie 2 6](#_Toc527929408)

[rendering classes dynamically 7](#_Toc527929409)

[Rendering lists 7](#_Toc527929410)

[Conditional rendering 8](#_Toc527929411)

[Optie 1 (if else) 8](#_Toc527929412)

[Optie 2 (enkel if) 8](#_Toc527929413)

[Handling events 9](#_Toc527929414)

[Binding event handlers 9](#_Toc527929415)

[Passing event arguments 10](#_Toc527929416)

[Nieuwe component toevoegen 10](#_Toc527929417)

[components gebruiken in andere components 10](#_Toc527929418)

[Meerdere van dezelfde component gebruiken in 1 component 11](#_Toc527929419)

[Optie 1 11](#_Toc527929420)

[Optie 2 (met list) 11](#_Toc527929421)

[Passing data to component 12](#_Toc527929422)

[In grote component met kleiner components erin 12](#_Toc527929423)

[In kleinere component die wordt opgeroepen in andere component 13](#_Toc527929424)

[tips 13](#_Toc527929425)

[Passing children 14](#_Toc527929426)

[In grote component met kleiner components erin 14](#_Toc527929427)

[In kleinere component die wordt opgeroepen in andere component 15](#_Toc527929428)

[Debugging react apps 15](#_Toc527929429)

[betekenis == $r 15](#_Toc527929430)

[Prop vs State 15](#_Toc527929431)

[Raising and handling events 16](#_Toc527929432)

[De regel 16](#_Toc527929433)

[Code 16](#_Toc527929434)

[Counters component 16](#_Toc527929435)

[Counter component 17](#_Toc527929436)

[Removing the Local State 18](#_Toc527929437)

[counter component (= controlled component) 18](#_Toc527929438)

[counters component (= parent component) 19](#_Toc527929439)

[Multiple Components in Sync 20](#_Toc527929440)

[Stateless Functional Components 20](#_Toc527929441)

[Destructuring Arguments 20](#_Toc527929442)

[Lifecycle Hooks 20](#_Toc527929443)

[Video tutorial 21](#_Toc527929444)

# React app aanmaken

C:\Users\Joren> create-react-app naam-van-app

# App runnen

C:\Users\Joren\naam-van-app> npm start

# Bootstrap in applicatie instalen

C:\Users\Joren\naam-van-app> npm i bootstrap

## In index.js toevoegen:

import ‘bootstrap/dist/css/bootstrap.css’;

# Component toevoegen

1. Onders src nieuwe map “components” maken
2. Onder src/components nieuw bestand maken: naamVanComponent.jsx

## In naamVanComponent.jsx

1. imrc tap (import react / component)
2. cc tap (create class)
3. geef naam aan class (bijvoorbeeld NaamVanComponent)

## Voeg toe aan index.js

import NaamVanComponent from ‘./components/naamVanComponent’;

ReactDom.render(<NaamVanComponent />, document.getElementById(“root”));

# 2 elementen meegeven aan component

class Counter extends Component{

render(){

return <React.Fragment><h1>Hey</h1><button>Knop</button> </React.Fragment>

}

}

!!! ALS WORDT GESAVED, WORDT TEKST HERVORMT TOT MEERDERE LIJNEN EN MOOIERE TEKST!!!

# Dynamisch tonen van value (in component)

Tussen { } kunnen alle expressies worden weergegeven

## Voorbeeld met statische tekst

class Counter extends Component{

state = {

count: 0

};

render(){

Return (

<React.Fragment>

<span>{this.state.count}</span>

<button>Knop</button>

</React.Fragment>

);

}

}

## Voorbeeld met wiskundige formule

class Counter extends Component{

render(){

Return (

<React.Fragment>

<span>{1 + 1}</span>

<button>Knop</button>

</React.Fragment>

);

}

}

## Voorbeeld met oproepen van functie die waarde terug geeft

class Counter extends Component{

state = {

count: 0

};

render(){

Return (

<React.Fragment>

<span>{this.formatCount()}</span>

<button>Knop</button>

</React.Fragment>

);

}

formatCount() {

const { count } = this.state;

return count === 0 ? ‘Zero’ : count;

}

# Setting Attributes

## Foto dynamisch instellen

class Counter extends Component{

state = {

imageUrl: ‘https://picsum.photos/200’

};

render(){

Return (

<div>

<img src={this.state.imageUrl} alt=””/>

</div>

);

}

}

## Bootstrap styling gebruiken/class gebruiken

class Counter extends Component{

render(){

Return (

<div>

<span className=”badge badge-primary”>zero</span>

</div>

);

}

}

***Tip: className=”badge badge-primary m-2” 🡺 geeft een margin van 2***

## Styles aanpassen optie 1

class Counter extends Component{

styles = {

fontSize: 50,

fontWeight: “bold”

};

render(){

Return (

<div>

<span style={this.styles} className=”badge badge-

primary”>zero</span>

</div>

);

}

}

## Styles aanpassen optie 2

class Counter extends Component{

render(){

Return (

<div>

<span style={{ fontSize: 30 }} className=”badge badge-

primary”>zero</span>

</div>

);

}

}

# rendering classes dynamically

class Counter extends Component{

state = {

count: 0

}

render(){

Return (

<div>

<span style={{ fontSize: 30 }} className=”this.getBadgeClasses”> {count} </span>

</div>

);

}

getBadgeClasses() {

let classes = “badge m-2 badge-”;

classes += (this.state.count ===0) ? “warning”: “primary”;

return classes;

}

}

# Rendering lists

class Counter extends Component{

state = {

tags: [‘tag1’, ‘tag2’, ‘tag3’]

}

render(){

Return (

<div>

<ul>

{ this.state.tags.map(tag => <li key={tag}>

{tag} </li>) }

</ul

</div>

);

}

}

# Conditional rendering

## Optie 1 (if else)

class Counter extends Component{

state = {

tags: [‘tag1’, ‘tag2’, ‘tag3’]

}

renderTags(){

if(this.state.tags.length === 0) return <p>There are no tags!</p>

return <ul> { this.state.tags.map(tag => <li key={tag}> {tag}

</li>) } </ul>

}

render(){

Return (

<div>

{ this.renderTags() }

</div>

);

}

}

## Optie 2 (enkel if)

class Counter extends Component{

state = {

tags: [‘tag1’, ‘tag2’, ‘tag3’]

}

render(){

Return (

<div>

{ this.state.tags.length === 0 && “please create a

new tag!”}

</div>

);

}

}

# Handling events

class Counter extends Component{

handleIncrement(){

console.log(‘Increment Clicked’);

}

render(){

Return (

<div>

<button onClick={this.handleIncrement}> Increment </button>

</div>

);

}

}

# Binding event handlers

class Counter extends Component{

stage = {

count:0

};

handleIncrement = () => {

this.setState({ count: this.state.count + 1 });

}

render(){

Return (

<div>

<button onClick={this.handleIncrement}> Increment </button>

</div>

);

}

}

# Passing event arguments

class Counter extends Component{

stage = {

count:0

};

handleIncrement = product => {

console.log(product)

this.setState({ count: this.state.count + 1 });

}

render(){

Return (

<div>

<button onClick={() => this.handleIncrement(product)}> Increment </button>

</div>

);

}

}

# Nieuwe component toevoegen

1. voeg een bestand toe aan de components folder
2. geef een naam eindigend met .jsx (bv. counters.jsx)
3. in bestand: imrc tap, cc tap

# components gebruiken in andere components

1. importeren van component in andere component

import Counter from ‘./counter’;

1. gebruiken van component

class Counters extend Component {

state = {};

render()

return (

<div>

<Counter />

</div>

);

}

}

# Meerdere van dezelfde component gebruiken in 1 component

!! elke component zal afzonderlijk werken en niet beïnvloed worden door de andere opgeroepen components !!

## Optie 1

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {}

render()

return (

<div>

<Counter />

<Counter />

<Counter />

</div>

);

}

}

## Optie 2 (met list)

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {

counters: [

{ id: 1, value: 0},

{ id: 2, value: 0},

{ id: 3, value: 0}

]

};

render()

return (

<div>

{this.state.counters.map(counter => <Counter key = {counter.id} />)}

</div>

);

}

}

# Passing data to component

## In grote component met kleiner components erin

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {

counters: [

{ id: 1, value: 4},

{ id: 2, value: 0},

{ id: 3, value: 0}

]

};

render()

return (

<div>

{this.state.counters.map(counter => <Counter key = {counter.id} value={counter.value} selected={true} />)}

</div>

);

}

}

## In kleinere component die wordt opgeroepen in andere component

class Counter extends Component {

state = {

value: this.props.value

};

handleIncrement = () => {

this.setState({ count: this.state.value +1});

};

render() {

return (

<div>

<span className={this.getBadgeClasses()}> {this.formatCount()}</span>

<button onClick =

{this.handleIncrement}

className=”btn btn-secondary brn-sm”

>

</div>

);

}

formatCount() {

const { value } = this.state;

return value === 0 ? ‘Zero’ : value;

}

}

## tips

**Om de meegegeven waarden te tonen in console, voeg in kleinere component (hier counter component) volgende lijn toe onder render():**

render(){

console.log(“props”, this.props);

return (

…

);

}

# Passing children

Een waarde doorgeven tussen de > en <.

## In grote component met kleiner components erin

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {

counters: [

{ id: 1, value: 4},

{ id: 2, value: 0},

{ id: 3, value: 0}

]

};

render()

return (

<div>

{this.state.counters.map(counter => <Counter key = {counter.id} value={counter.value}/>

<h4>Counter #{counter.id}</h4>

</Counter>

</div>

);

}

}

## In kleinere component die wordt opgeroepen in andere component

class Counter extends Component {

state = {

value: this.props.value

};

handleIncrement = () => {

this.setState({ count: this.state.value +1});

};

render() {

return (

{this.props.children}

<div>

<span className={this.getBadgeClasses()}> {this.formatCount()}</span>

<button onClick =

{this.handleIncrement}

className=”btn btn-secondary brn-sm”

>

</div>

);

}

formatCount() {

const { value } = this.state;

return value === 0 ? ‘Zero’ : value;

}

}

# Debugging react apps

1. installeer React Developer Tools in Chrome of Firefox
2. ga naar debugging en bij meer staat nu react

## betekenis == $r

in console, type $r en ge krijgt de aangeduide component

# Prop vs State

Props = data die we aan een component geven, ze zijn read-only dus we kunnen

ze niet aanpassen

State = data die lokaal/private is van een component, andere components kunnen

niet aan deze data

# Raising and handling events

## De regel

De component die een stuk van de state bezit, moet de gene zijn die dit aanpast.

## Code

### Counters component

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {

counters: [

{ id: 1, value: 4},

{ id: 2, value: 0},

{ id: 3, value: 0}

]

};

handleDelete = () => {

console.log(“Event Handler Called”)

};

render()

return (

<div>

{this.state.counters.map(counter => <Counter key = {counter.id} onDelete={this.handleDelete} value={counter.value}/>

<h4>Counter #{counter.id}</h4>

</Counter>

</div>

);

}

}

### Counter component

class Counter extends Component {

state = {

value: this.props.value

};

handleIncrement = () => {

this.setState({ count: this.state.value +1});

};

render() {

return (

{this.props.children}

<div>

<span className={this.getBadgeClasses()}> {this.formatCount()}</span>

<button onClick =

{this.handleIncrement}

className=”btn btn-secondary brn-sm”

> Increment </button>

<button onClick =

{this.props.onDelete}

className=”btn btn-secondary brn-sm”

> Delete </button>

</div>

);

}

formatCount() {

const { value } = this.state;

return value === 0 ? ‘Zero’ : value;

}

}

# Removing the Local State

Controlled component = component dat zelf geen lokale data heeft maar al zijn

data krijgt van zijn parent component (heeft geen state)

## counter component (= controlled component)

class Counter extends Component {

render() {

return (

{this.props.children}

<div>

<span className={this.getBadgeClasses()}> {this.formatCount()}</span>

<button onClick =

{() => this.props.onIncrement( this.props.counter)}

className=”btn btn-secondary brn-sm”

> Increment </button>

<button onClick =

{this.props.onDelete}

className=”btn btn-secondary brn-sm”

> Delete </button>

</div>

);

}

getBadgeClasses() {

let classes = “badge m-2 badge-“;

classes += this.props.counter.value === 0 ? “warning” : “primary”;

return classes;

}

formatCounter() {

const { value } = this.props.counter;

return value === 0 ? “zero” : value;

}

}

## counters component (= parent component)

import react, { Component } from “react”;

import Counter from “./counter”

class Counters extends Component {

state = {

counters: [

{ id: 1, value: 4},

{ id: 2, value: 0},

{ id: 3, value: 0}

]

};

handleIncrement = counter => {

const counters = […this.state.counters];

const index = counters.indexOf(counter);

counters[index] = {…counter};

counters[index].value++;

this.setState({ counters })

};

handleReset = () {…

};

handleDelete = () => {…

};

render()

return (

<div>

<button

onClick={this.handleReset}

>

Reset

</button>

{this.state.counters.map(counter => (

<Counter

key = {counter.id} onDelete={this.handleDelete}

onIncrement={this.handleIncrement}

counter = {counter}

/>

))}

</div>

);

}

}

# Multiple Components in Sync

Zie video 1:54:43

<https://youtu.be/Ke90Tje7VS0?t=6883>

# Stateless Functional Components

Andere optie: gebruik een constant ipv een class voor de html aan mee te geven

<https://youtu.be/Ke90Tje7VS0?t=7577>

# Destructuring Arguments

**Ipv:** this.props.onReset en this.props.counters.map(…)

**Boven return(…) en onder render() {…}:**

const {onReset, counters}

**this.props.onReset en this.props.counters.map(…) vervangen door:**

onReset en counters.map(…)

Voor stateless functional components:

<https://youtu.be/Ke90Tje7VS0>

# Lifecycle Hooks

<https://youtu.be/Ke90Tje7VS0?t=7854>

lifecycle hooks = methods om op een bepaald moment in onze dom te springen,

react roept met deze methods automatisch dit moment op.

verschillende lifecycle fases:

1. Mount = instant van component wordt gemaakt en geïnjecteerd in de dom
   1. constructor
   2. render
   3. componentDidMount
2. Update = wanneer de state of props van een component veranderen
   1. render
   2. componetnDidUpdate
3. Unmount = wanneer component wordt verwijderd van de dom (zoals delete

van shoppingcart item)

* 1. componentWillUnmount

voor werking/gebruik van phases: <https://youtu.be/Ke90Tje7VS0>

# Video tutorial

<https://www.youtube.com/watch?v=Ke90Tje7VS0>