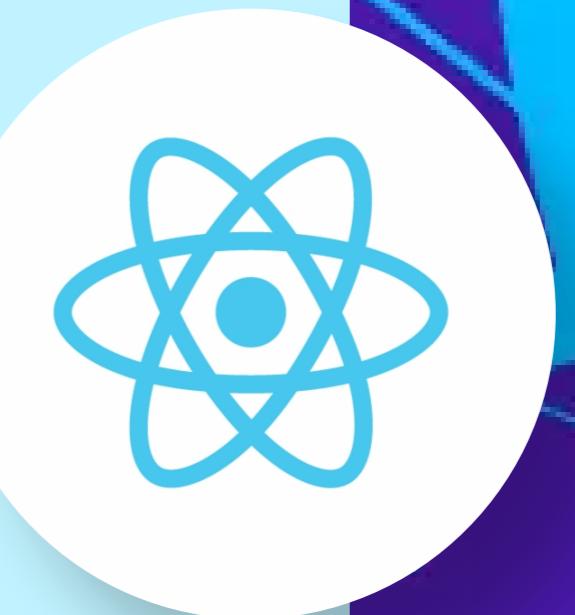


REACT basics

Les 19:

# React Hooks



# Onderwerpen les:

- Wat zijn React hooks?
- useState
- useEffect

# React hooks

## Wat zijn React hooks?

Hooks geven ‘function components’ toegang tot de ‘state’ en andere React-functies.

Hierdoor zijn ‘class components’ over het algemeen niet meer nodig.

React heeft een aantal **ingebouwde hooks** zoals useState en useEffect.

Bekijk hier alle React Hooks

| <https://reactjs.org/docs/hooks-reference.html>

Je kunt ook je eigen hooks maken om ‘stateful’ gedrag tussen verschillende componenten te hergebruiken. Dit gaan we deze les niet behandelen.

| <https://reactjs.org/docs/hooks-custom.html>

# React hooks

## useState

useState is een Hook waarmee je de **React-state** kunt toevoegen aan functiecomponenten.

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```
import React, { useState } from 'react';  
  
function Example() {  
  // Declare a new state variable, which we'll call "count"  
  const [count, setCount] = useState(0);  
}
```

Class component state

vs

Function component state

| <https://reactjs.org/docs/hooks-state.html>

# React hooks

## useState

### Wat doet het aanroepen van useState?

Het declareert een **"state variable"**. In het voorbeeld wordt de variabele count genoemd, maar dit had ook jojo kunnen zijn. Dit is een manier om sommige waarden te "behouden" tussen de functieaanroepen – useState is een nieuwe manier om exact dezelfde mogelijkheden te gebruiken die this.state biedt in een klasse.

### Wat geven we als argument mee aan useState()?

Het enige argument voor de hook useState() is de **initiële state**. We kunnen een nummer of een string meegeven als dat alles is wat we nodig hebben. In ons voorbeeld willen we alleen een getal voor het aantal keren dat de gebruiker heeft geklikt, dus geven we 0 door als beginstatus .

```
function ExampleWithManyStates() {  
  // Declare multiple state variables!  
  const [age, setAge] = useState(42);  
  const [fruit, setFruit] = useState('banana');  
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

# React hooks

## useState

### Wat geeft useState terug (return)?

Het returns een paar waarden: **de huidige state en een functie die deze bijwerkt**. Daarom schrijven we const [count, setCount] = useState(). Dit is vergelijkbaar met this.state.count en this.setState in een klasse, behalve dat je ze in een paar krijgt.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

### Wat doet ons voorbeeld?

We declareren een state variabele met de naam count en stellen deze in op 0. React onthoudt de huidige waarde tussen opnieuw renderen en geeft de meest recente aan onze functie. Als we de huidige telling willen bijwerken, kunnen we setCount aanroepen.

# React hooks

## useState

### Lezen van State

Als we het huidige aantal in een class componenten willen weergeven, gebruiken we `this.state.count`. In een function component kunnen we `count` direct gebruiken.

```
<p>You clicked {this.state.count} times</p>
```

```
<p>You clicked {count} times</p>
```

### Updaten van state

In een class component gebruiken we `this.setState()` om het aantal te wijzigen. In een function kunnen we `setCount` aanroepen met `count` als variabele.

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>  
  Click me  
</button>
```

```
<button onClick={() => setCount(count + 1)}>  
  Click me  
</button>
```

# React hooks

## useEffect

met useEffect vertel je React dat je component iets moet doen na renderen. React onthoudt de functie die je hebt doorgegeven ("effect") en roept deze later aan na het uitvoeren van de DOM-updates. In het voorbeeld hebben we de titel van het document ingesteld, maar we kunnen ook gegevens ophalen of een andere API aanroepen.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

DidMount, DidUpdate

VS

useEffect

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

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

| <https://reactjs.org/docs/hooks-effect.html>

# React hooks

## useState

### Prestaties optimaliseren door effecten over te slaan

In sommige gevallen kan het opruimen of toepassen van het effect na elke render een prestatieprobleem veroorzaken. In class components kunnen we dit oplossen door een extra vergelijking te schrijven met prevProps of prevState binnen componentDidUpdate.

Deze vereiste is zo algemeen dat deze is ingebouwd in de useEffect Hook API. U kunt React vertellen om het toepassen van een effect over te slaan als bepaalde waarden niet zijn veranderd tussen opnieuw renderen. Om dit te doen, geeft u een array door als een optioneel tweede argument voor useEffect.

```
componentDidUpdate(prevProps, prevState) {
  if (prevState.count !== this.state.count) {
    document.title = `You clicked ${this.state.count} times`;
  }
}
```

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]); // Only re-run the effect if count changes
```

| <https://www.youtube.com/watch?v=0ZJgljluY7U>