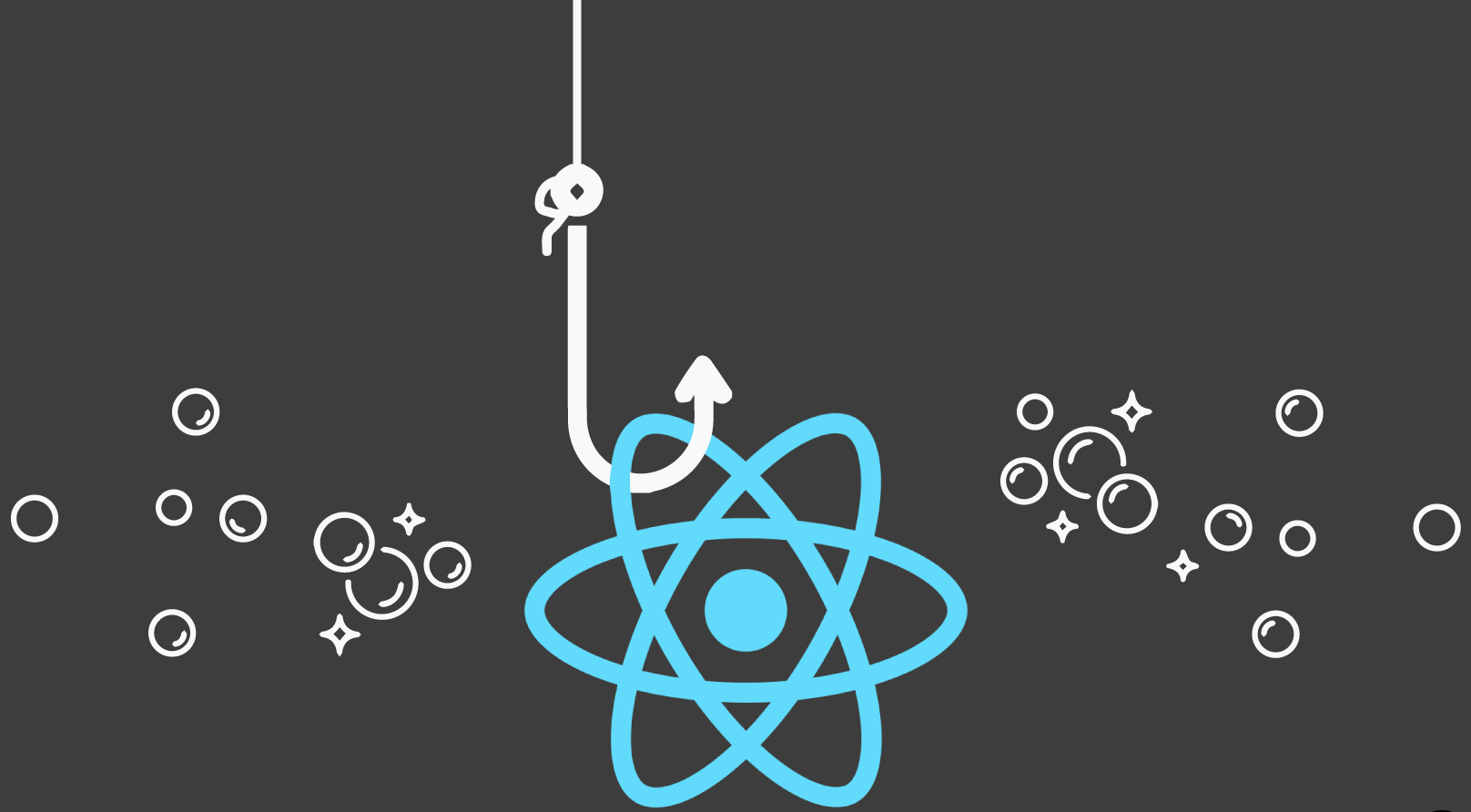
Web 3

React Hooks

HO
GENT


HO
GENT

Gebruikte technologieën

- useState
- useEffect
- useRef
- useContext
- React Hooks

HO
GENT

Hooks

- Hooks zijn **speciale functies** die toelaten om “**hook into**” **React features**. Laten ons toe om functie components te gebruiken voor React functies waarvoor we anders een class component nodig hadden
- Verschillende **standaard hooks** in React
  - **useState**
  - **useEffect**
  - **useContext**
  - **useReducer**
  - …

https://beta.reactjs.org/reference/react

HO GENT

State

- Tot nu toe hebben we een component tree gebouwd waarbij **data** van **parent** naar **child** kon gaan door middel van **props**. Deze data was niet aan te passen
- De **state** van een React applicatie is gedreven door **data** die de **mogelijkheid** heeft om te **wijzigen**. Bijvoorbeeld nieuwe data aanmaken, bestaande wijzigen, of bestaande data verwijderen
- De **state** en de **props** hebben een **relatie** met **elkaar**. Als de **state** van een **component wijzigt**, **wijzigen** ook de **props**

State (2)

- **Geen kwestie** om gewoon een **variabele aan maken** in het begin van onze component om een state te hebben. Anders **manueel re-renderen** van de **pagina's** (niet aan te raden)
- Vroeger werd de state bijgehouden enkel maar in statefull components (classes)
- Tegenwoordig door gebruik te maken van **Hooks** ook mogelijk in functie components door **useState**, **useReducer**

HO
GENT

useState

- Deze hook zorgt er voor dat we **state bij** kunnen **houden** in onze **applicatie**
- Wordt gebruikt in **functie components**
- React zal deze **state bijhouden** tussen **twee re-renders**
- De **useState** hook geeft een **array** terug met twee elementen in:
  - De **huidige state**
  - Een **functie die toelaat om de state up te daten**.

https://reactjs.org/docs/hooks-state.html

HO GENT

useState (2)

- **Array destructuring** om deze hook te gebruiken

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

const [counter, setCounter] = useState(0);
```

Huidige state

Functie om state up te daten

Hook

Initiële waarde

HO GENT

useEffect

- Deze **hook** zorgt er voor dat we **side effecten** kunnen **uitvoeren** in functie componenten
- Voorbeelden van side effecten zijn het **ophalen** van **data** van een **API**, **manueel aanpassen** van de **DOM** in React components, ...
- **Effecten** worden **uitgevoerd** bij iedere **re-render**, kan **gelimiteerd** worden met **skip variabele**

https://reactjs.org/docs/hooks-effect.html

useEffect (2)

- Gebruik maken van de useEffect hook

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

useEffect(() => {
   console.log("Every re-render of data");
});

useEffect(() => {
   console.log("Every refresh of the page");
}, []);

useEffect(() => {
   console.log("Everytime counter changes");
}, [counter]);
```

HO GENT

useEffect - Without Cleanup

- Soms willen we nog code uitvoeren nadat React de DOM heeft gewijzigd. Het uitvoeren van een **netwerk request**, **manuele DOM manipulaties**, en bijvoorbeeld **logging**. Bij deze effecten moeten we **geen cleanup** doen. Omdat we deze kunnen **uitvoeren** en **onmiddellijk vergeten**

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

useEffect(() => {
  console.log("Every re-render");
});
```

HO GENT

useEffect - With Cleanup

- Soms willen we nog code uitvoeren nadat React de DOM heeft gewijzigd. Bijvoorbeeld het **instellen** van een **interval** waarbij we er willen voor zorgen dat de **interval verwijderd** wordt als de **component niet meer gebruikt** wordt. Daarvoor **gebruiken** we een **cleanup**

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

useEffect(() => {
   const interval = setInterval(() => {
       console.log('This will run every second');}, 1000);
   return () => clearInterval(interval);
}, []);
```

HO GENT

Hooks - REGELS

- Het **aanroepen** van een **hook** enkel maar **at the top level**. Dus **geen hooks aanroepen** in een **loop**, een **conditie**, of een **andere functie**.

```
if (name !== '') {
    useEffect(() => {
          localStorage.setItem('formData', name);
    });
}

useEffect(() => {
    if(name !== '') {
          localStorage.setItem('formData', name);
    });
}
```

https://reactjs.org/docs/hooks-rules.html

HO GENT

useRef

- In React maken we **geen gebruik** van **`getElementById`** of **andere selectoren** om bvb. JavaScript methodes op op te roepen
- Maar we kunnen wel de tegenhanger in React gebruiken aan de hand van de **useRef hook**
- De **useRef** hook gaat een **soort** van **referentie maken** naar een element waar we later bvb. een **methode** willen **op oproepen** of een **property uithalen**
- Wordt veel gebruikt voor de bvb. de focus op een inputelement te gebruiken

https://reactjs.org/docs/hooks-reference.html#useref

useRef (2)

- Belangrijk hierbij is dat de **properties** of de **methodes** van een element beschikbaar zijn in de **`current` property** van **deze referentie** (zie voorbeeldcode)

```
import React, { useRef } from ‘react’;

const inputRef = useRef(null);

<input type=“text” ref={inputRef} />

<button onClick={() => inputRef.current.focus()}>Focus
me</button>
```

HO
GENT

useContext

- Als we **data** wouden **doorgeven** (van Parent naar Child) maakten we gebruik van **props** die we **doorgaven**
- Als we vanuit een Parent component data wouden doorgeven naar bvb. de vijfde child component dan moesten we dit altijd via de props doen (**redelijk complex**)
- Daarom kunnen we gebruik maken van de **useContext** hook in React om data **automatisch door** te **kunnen geven** naar **alle Child components** (maakt niet uit hoe diep)

https://reactjs.org/docs/hooks-reference.html#usecontext

useContext (2)

- We gebruiken dit door **eerst** een **Context aan** te **maken** en dan met een **Provider** deze ter **beschikking** te **stellen** in een Parent component

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

const themes = { light: { background: 'yellow'}, dark: {
                        background: 'red'}};

const ThemeContext = createContext(themes.light);
```

HO GENT

useContext (3)

- Dan gebruiken we de ThemeContext.**Provider** in de Parent Component

```
import React from 'react';

export const Component = () => {

   return (
        <ThemeContext.Provider value={themes.dark}>
            <Child />
        </ThemeContext>
)};
```

HO GENT

useContext (4)

- Dan gebruiken we de **useContext** om de **values** hier uit te halen

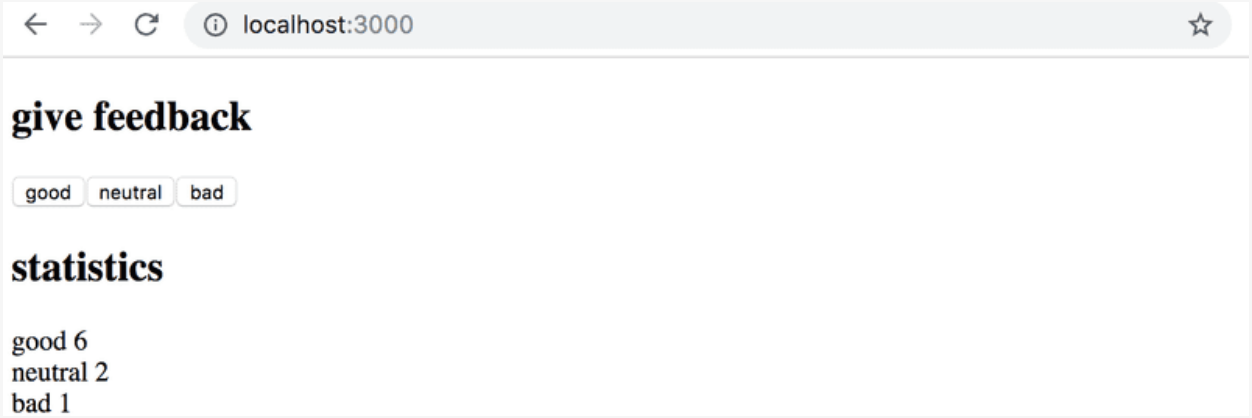
```
import React, { useContext } from 'react';

export const SecondChild = () => {

   const theme = useContext(ThemeContext);

   return (
            <p style={{ backgroundColor:
                theme.background}}>Test</p>
)};
```

OEFENING 1

- Maak een React applicatie met een Feedback component aan. De feedback component bestaat uit een pagina met drie knoppen om feedback te geven; **GOED**, **NEUTRAAL**, **SLECHT**
- Gebruik de useState om de state bij te houden van hoeveel keer er al op een feedback knop is gedrukt geweest (TIP: 3 hooks nodig hiervoor)
- Laat het aantal kliks zien per knop zoals op de afbeelding
- Je mag de app zelf een styling geven hoe je wil

HO
GENT

OEFENING 1



HO GENT