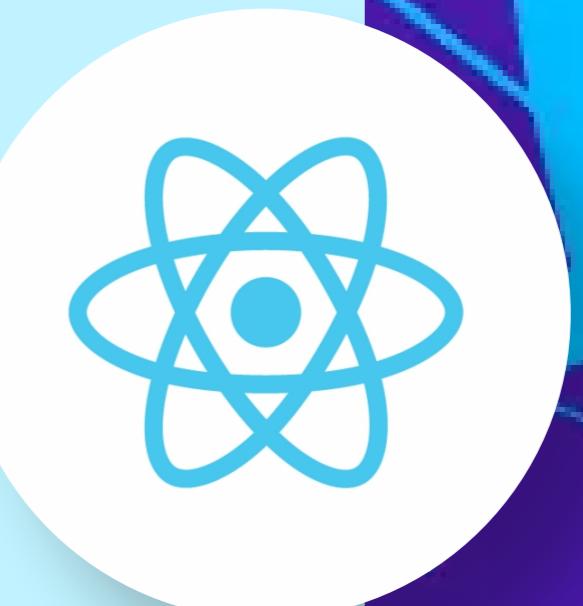


REACT basics

Les 15:

# **JavaScript classes, class components, state & events**



# Onderwerpen les:

- Classes en objecten in JavaScript
- Class components
- State
- Events
- Create React app
- Modulaire code / data flow

# Leerdoelen

- ✓ Aan het einde van de dag kun je een class component maken
- ✓ Aan het einde van de dag kun je state toepassen in components
- ✓ Aan het einde van de dag kan je een event afhandelen
- ✓ Aan het einde van de dag weet je wat modulair coderen is

# Classes en objecten in javascript

## Wat is een class?

Een class is de blauwdruk van objecten dus niet het object zelf. In een class staan variabelen en methodes. Als je dit vergelijkt met iets uit de echte wereld:

Je maakt aan de hand van een blauwdruk een huis. De blauwdruk is de class. In deze blauwdruk staan variabelen als aantal ramen en deuren, vierkante meter, kleur, huisnummer, materiaal etc. Sommige van deze variabelen zijn van te voren ingevuld zoals de vierkante meters aantal ramen etc. Deze variabelen zullen dus voor elk huis dat je maakt aan de hand van de blauwdruk hetzelfde zijn. De kleur en het huisnummer staan niet vast en zullen dus per huis kunnen verschillen of zelfs uniek zijn.



# Classes en objecten in javascript

## Wat is een object?

Een object is een **instantie van een class**. Dorpstraat 12 is een instantie van de class huis. Er kunnen dus meerdere objecten worden gemaakt van dezelfde class.

## Wat is een constructor

Een constructor is een **speciale functie in de class die objecten creëert en initialiseert van de class**.

In javascript wordt de constructor aangeroepen wanneer er een object wordt aange- maakt met het **'new' keyword**.

In de constructor bepaal je de values voor variabelen van de instantie van de class.

# Classes en objecten in javascript

```
1 class Car {  
2     constructor(merk, kleur, jaar = 2022) {  
3         this.merk = merk;  
4         this.kleur = kleur;  
5         this.jaar = jaar;  
6     }  
7 }  
8  
9 let myCar = new Car("Mercedes", "rood");  
10 console.log(myCar);  
11 console.log(myCar.merk);  
12 console.log(myCar.jaar);
```

# Class components

We hebben in de vorige lessen components gemaakt met een function of een arrow function. Een tweede manier is het gebruiken van classes.

## Class components vs function components

- Function als je een component alleen input via props geeft en een ui rendert.  
Dan is het een stateless functional component.
- Een class component gebruik je wanneer je state gaat gebruiken.

# Class components

## Maken class component

Een class component extends de class React.Component zodat deze de functionaliteiten over kan nemen die al voor je zijn geschreven in React.

```
37 // een react class component extends React.Component. Maakt een subclass van React.Component
38 class Counter extends React.Component {
39     constructor(props) {
40         super() // aanroepen constructor van de React.Component
41         this.score = props.score
42     }
43
44     // Een react class heeft een render nodig om de return react class
45     render() {
46         return (
47             <div className="counter">
48                 <button className="counter-action decrement"> - </button>
49                 <span className="counter-score">{this.score}</span>
50                 <button className="counter-action increment"> + </button>
51             </div>
52         )
53     }
54 }
```

# State in React

Props (properties) en state hebben informatie dat invloed uitoefent op de manier waarop output wordt gerenderd. Het verschil is dat props worden meegegeven aan een component (zoals parameters bij een functie) En **state wordt juist binnen in de component gemanaged** (zoals een variabele in een functie).

De state is dus de staat waarin een component zich bevindt. Oftewel de waarde van een variabele zoals de score van onze Counter component.

```
39  class Counter extends React.Component {  
40    constructor() {  
41      super()  
42      this.state = { // in this.state wordt de staat van een component bepaald en bijgehouden  
43        score: 0  
44      };  
45    }  
46  
47    //ophalen state : this.state.x  
48    render() {  
49      return (  
50        <div className="counter">  
51          <button className="counter-action decrement"> - </button>  
52          <span className="counter-score">{this.state.score}</span>  
53          <button className="counter-action increment"> + </button>  
54        </div>  
55      )  
56    }  
57  }
```

# Class components

## State: een kortere manier van noteren

Er is een shortcut in React om de constructor kortschrijven te noteren. Dit werkt niet in alle browsers, maar omdat wij babel script zelf inladen werkt het wel overal.

```
39  class Counter extends React.Component {
40    // constructor() {
41    //   super()
42    //   this.state = { // in this.state wordt de staat van een component bepaald en bijgehouden
43    //     score: 0
44    //   };
45  }
46
47  state = {
48    score: 0
49  };
50
```

# Events

## this.setState

Met this.setState metod kun je de state van een component aanpassen. Bijvoorbeeld de score 1 omhoog of omlaag zetten. Om dit te koppelen aan een event, zoals een click event kunnen we de setState in een functie zetten die wordt aangeroepen na het klikken.

### twee manieren

1 aanroepen method met `onclick={this.decrementScore.bind(this)}`

Doordat onze class een andere class extend hebben onze methods geen toegang tot this. Onze render waar we de method aanroepen is er wel toegang tot this, daarom geven we this mee via een bind().

2 aanroepen method met `onclick={this.decrementScore}`, maar de method is een arrow function.

Een arrow function is automatisch gebonden aan de scope waarin ze zijn geschreven en hebben daarom wel toegang tot this.

# Events

```
50
51 // method die wordt aangeroepen (kan niet bij this)
52 decrementScore() {
53     console.log("hoi vanuit decrement score");
54     this.setState({
55         score: this.state.score - 1
56     })
57 }
58
59 // onclick event met bind this zodat method erbij kan komen
60 render() {
61     return (
62         <div className="counter">
63             <button className="counter-action decrement" onClick={this.decrementScore.bind(this)}> - </button>
64             <span className="counter-score">{this.state.score}</span>
65             <button className="counter-action increment"> + </button>
66         </div>
67     )
68 }
```

1: met bind

# Events

```
59 // method als arrow function
60 decrementScore = () => {
61     console.log("hoi vanuit increment score");
62     this.setState({
63         score: this.state.score + 1
64     })
65 }
66
67 // onclick event zonder bind
68 render() {
69     return (
70         <div className="counter">
71             <button className="counter-action decrement" onClick={this.decrementScore.bind(this)}> - </button>
72             <span className="counter-score">{this.state.score}</span>
73             <button className="counter-action increment"> + </button>
74         </div>
75     )
76 }
```

## 2: zonder bind met arrow function

# State based on previous state

## Asynchroon wat is dat?

Synchroon uitvoeren van taken in programma's is 1 voor 1. Een taak begint pas als de andere is afgelopen.

Asynchroon uitvoeren van taken is een **volgende taak al starten terwijl een andere taak of taken nog niet zijn afgerond**.

### Synchroon taken in huis

Koffie zetten en wachten tot het klaar is

Koffie opdrinken

Wasje draaien en wachten tot klaar is

Was ophangen

Kamer opruimen

### Asynchroon taken in huis

Koffie apparaat aanzetten

Wasmachine aanzetten

Koffie is klaar opdrinken

Kamer opruimen

Was ophangen

# State based on previous state

## Gevolgen voor onze app

Doordat React onze state veranderingen Asynchroon verwerkt (de app blijft doorlopen als de state moet veranderen) kunnen een groot aantal setState calls gemerged worden en tegelijkertijd worden uitgevoerd. Dit zal bij onze kleine app niet voor problemen zorgen, maar er is een betere manier om onze score bij te houden. met prevState wordt eerst gekeken naar de voorgaande staat van de component.

```
59 // method met prevState
60 incrementScore = () => {
61   console.log("hoi vanuit increment score");
62   this.setState( prevState => ({
63     score: prevState.score + 1
64   }));
65 }
```

# State toepassen

## Spelers als state

We gaan onze app nog iets verder uitbreiden door de spelers om te zetten naar state ipv props en vervolgens een methode maken om de spelers te verwijderen.

- 1 verander de App function component naar een class component
- 2 maak een state object en verwijst naar state ipv props in render
- 3 verwijder de players array
- 4 geef geen props meer mee aan App

# State toe passen

## Spelers als state

```
77  class App extends React.Component {  
78    state = {  
79      players: [  
80        {  
81          name: "Player 1",  
82          id: 1  
83        },  
84        {  
85          name: "Player 2",  
86          id: 2  
87        }  
88      ]  
89    }  
90    render() {  
91      return (  
92        <div className="scoreboard">  
93          <Header title="Scoreboard" totalPlayers={this.state.players.length}/>  
94          {this.state.players.map( player =>  
95            <Player  
96              name={player.name}  
97                key={player.id.toString()}  
98            />  
99          )}  
100     )  
101   )  
102 }
```

# State toepassen

## Remove player

In deze stap geven we de app de mogelijkheid om een speler te verwijderen.

5 maak een remove player function

6 voeg een prop id en remove player toe

7 voeg een delete button toe

# State toepassen

## Remove player

```
92  handleRemovePlayer = (id) => {
93    this.setState( prevState => {
94      return {
95        players: prevState.players.filter( p=> p.id !== id ) //p staat voor current item.
96          // zoekt naar een p.id die niet overeenkomt met de parameter id
97      };
98    });
99  }
```

# State toepassen

## Remove player

```
101 // toevoegen id en removePlayer prop
102 render() {
103   return (
104     <div className="scoreboard">
105       <Header title="Scoreboard" totalPlayers={this.state.players.length}/>
106       {this.state.players.map( player =>
107         <Player
108           name={player.name}
109           id={player.id}
110           key={player.id.toString()}
111           removePlayer={this.handleRemovePlayer}
112         />
113       )}
114     </div>
115   )
116 }
117 }
```

# State toepassen

## Remove player

```
28 // button met onClick anonieme functie. toegang tot functie in andere Component via props
29 const Player = (props) => {
30   return (
31     <div className="player">
32       <span className="player-name">
33         <button className="remove-player" onClick={() => props.removePlayer(props.id)}>x</button>
34         {props.name}
35       </span>
36       <Counter />
37     </div>
38   )
39 }
```

# Create React App

## Installeren en gebruiken in project

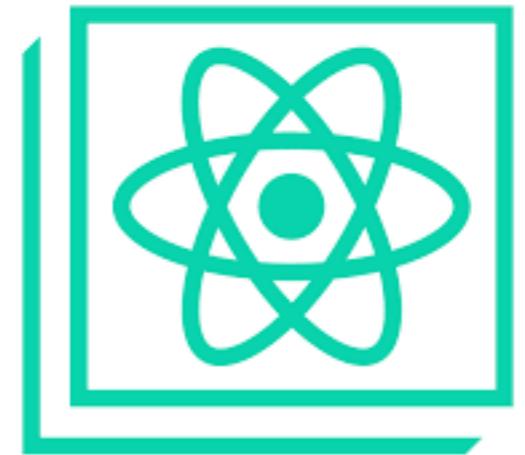
Create React App maakt een development omgeving zodat je de laatste JavaScript features kan gebruiken met React. Het geeft een goede developer ervaring, en optimaliseert je app voor productie. Je hebt nodig:

### Benodigdheden

Wat?	Check versie in terminal / prompt	download
Node >= 14.0.0	node-v	<a href="https://nodejs.org/en/download/">https://nodejs.org/en/download/</a>
NPM >= 5.6	npm-v	<a href="https://docs.npmjs.com/downloading-and-installing-node-js-and-npm">https://docs.npmjs.com/downloading-and-installing-node-js-and-npm</a>

### Create React App stappen

Wat?	hoe?
Installeer npm op project	1 ga naar juiste map in terminal / prompt 2 typ: npm install
Installeren React	1 ga naar juiste map in terminal / prompt 2 typ: npx create-react-app my-app (verander my app in gewenste naam)
Runnen app	1 ga naar de map my-app (naam app) in terminal / prompt 2 typ: npm start



# Modulaire code

Modulaire code is code dat is gesplitst in segmenten (modules), elk bestand is verantwoordelijk voor een feature of specifieke functionaliteit. Dit wordt gedaan om verschillende redenen:

## **Single responsibility principle:**

Goede code is leesbaar, daarom wordt verwacht dat een module 1 taak. Een module voegt 1 functionaliteit of 1 feature toe aan de applicatie

## **Makkelijker naviigeerbaar:**

Modules die opgesplitst zijn en een duidelijke naam hebben zijn makkelijker leesbaar voor developers.

## **Makkelijker debuggen:**

Bugs zijn sneller te vinden in kleine opgesplitste code.

## **Nettere code en DRY:**

DRY = Dont repeat yourself. Modules zijn makkelijk te hergebruiken.

# Modulaire code

## Toepassen modulaire code

De code in het JavaScript bestand bestaat uit meerde componenten. Deze componenten behoren een eigen module te zijn en moeten daarom in een eigen js bestand geplaatst worden.

### Stap 1: Opknippen code

Knip de code in losse modules

### Stap 2: Importeer dependancies

Bovenaan in de module importeer je de dependancies (alles wat je nodig hebt in de module)

```
1 import React from 'react';
2 import Counter from './Counter';
```

# Modulaire code

## Stap 3: Exporteer component

De component gemaakt binnen de module wil je ergens anders gebruiken. Om dit werkend te krijgen zul je onderaan in de module de component moeten exporteren. Indien een component in een module niet wordt geexporteerd is de component alleen beschikbaar binnen zijn eigen module.

```
37 export default Counter;
```

# Modulaire code

## Toepassen modulaire code op het project

### Header module

- 1 import react
- 2 kopieer en plak de Header component
- 3 exporteer component
- 4 import header component in App

### Player module

- 1 import react
- 2 kopieer en plak de Player component
- 3 exporteer component

### Counter module

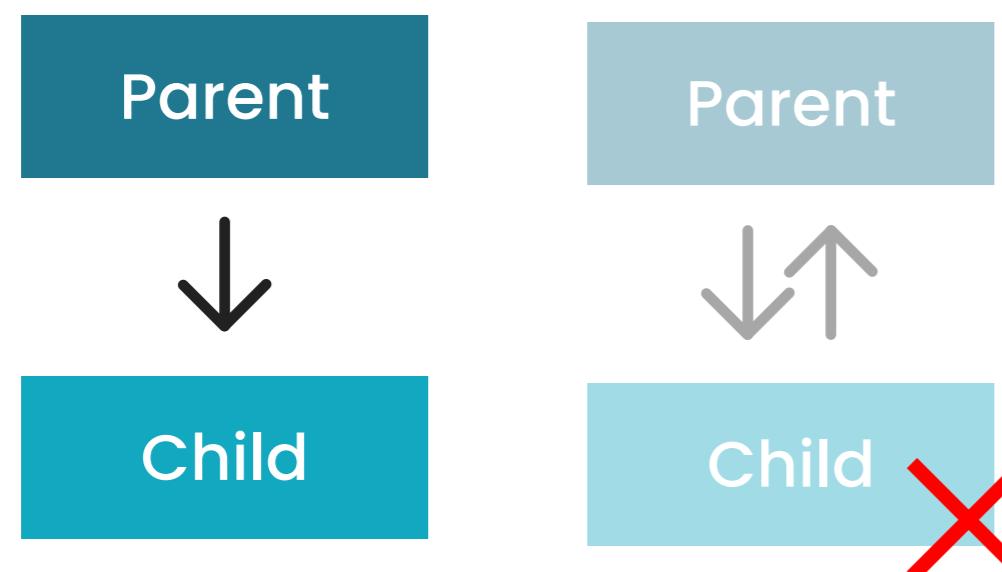
- 1 import react
- 2 kopieer en plak de Counter component
- 3 exporteer component
- 4 importeer Counter in Player

# State & Data flow

## unidirectional data flow

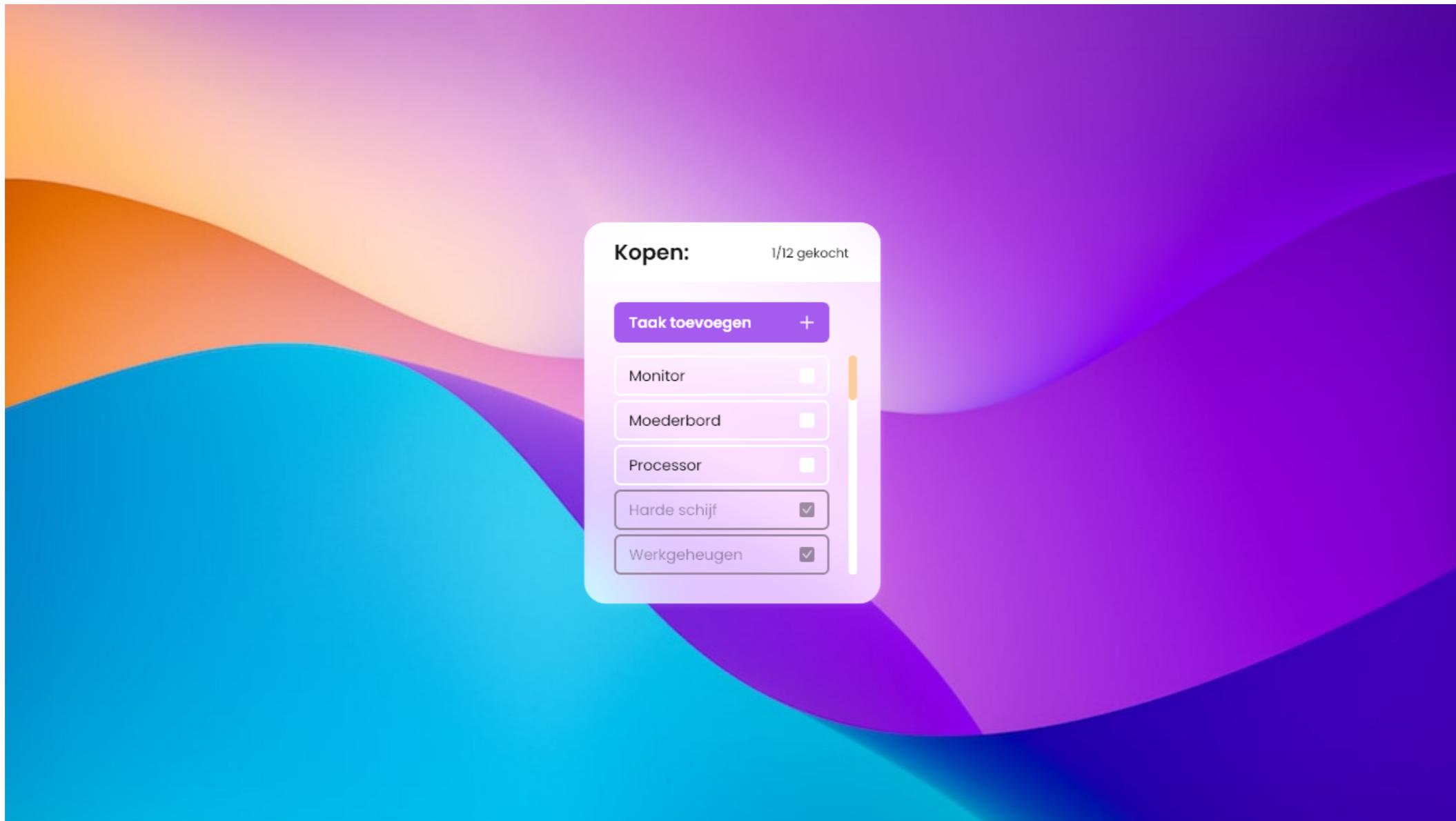
Eenrichtingsgegevensstroom waarbij de gegevens slechts in één pad kunnen worden verplaatst wanneer ze tussen verschillende delen van het programma worden overdragen. Je kunt alleen gegevens overdragen van ouder naar kind en niet andersom.

Dit betekent dat de onderliggende componenten de gegevens niet zelf kunnen bijwerken of wijzigen, zodat een nette gegevensstroomarchitectuur wordt gevuld. Dit betekent ook dat je de datastream beter kunt beheersen.



# Opdracht

**Maak een boodschappenlijstje**



# Opdracht

## Maak een boodschappenlijstje

### Wat kan de app? (basis)

- 1 Boodschappen toevoegen aan een lijst
- 2 Boodschappen verwijderen
- 3 Boodschappen markeren als gekocht

### Wat kan de app? (extra)

- 1 Bijhouden hoeveel boodschappen totaal en hoeveel gekocht
- 2 Gekochte boodschappen onderaan plaatsen

### Wat kan de app? (optioneel)

- 1 Datum en tijd toevoegen
- 2 Boodschappen aanpassen nadat ze zijn gemaakt