REACT

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library[3] for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies.

Aan het eind van deze lessenserie heb je database-driven quiz app gemaakt.

- Inleiding
- *** Deel 1 ***
- Installatie
- JSX en state variabelen
- State variabelen gebruiken
- Quiz (vragen)
- *** Deel 2 Inleiding ***
- Quiz App - deel 1
- Quiz App - deel 2
- *** Deel 3 REST API Inleiding***
- REST API 1
- REST API 2
- Installatie React packages

Inleiding

React

React is een framework of library (een verzameling van tools) waarmee je een userinterface kan maken. React is dus een front end taal.

Q: PHP en Yii zijn back-end talen. Weet je het verschil nog?

> image-1657984534973.png
>
> Juist! Front-end fraai je op je eigen PC/Laptop in de browser en Back-end draait op een server en stuurt de resultaten naar jouw browser.

Q; PHP draait op de server? Hmm mijn PHP draait toch op XAMPP op mijn Laptop?

> Ja dat is zo. Omdat wij niet allemaal een server hebben, gebruiken wij XAMPP. XAMPP (of soortgelijke programma's) doet alsof zij een server zijn. Dat heet emulatie. XAMPP emuleert dus een server.

De meeste web-applicaties zijn een combinatie van front-end en back-end componenten. Wij zullen in deze module ook een combinatie gebuiken.

Wij gaan een quiz applicatie maken op de front-end met REACT en later gaan we een back-end maken waarin we de quizvragen kunnen bijhouden. Via een REST API gaan we dan vanuit REACT de back-end server benaderen. De backend maken we in Yii/PHP.

Een REST API is een standaard manier waarop een programmagegevens kan uitwisselen met een ander programma. Het word heel veel gebruikt in de webwereld.

Wat ga je leren?

Wat je gaat leren is een introductie in REACT waarbij we een volledig werkende applicatie gaan maken.

Daarnaast gaan we leren hoe we via een REST API gegevens uit een Yii/PHP applicatie kunnen halen.

Wat moet je weten?

Je kunt omgaan met Visual Studio Code en de terminal en je weet hoe je software kan installeren. Je weet hoe je in de VCS terminal commando's kan geven. Dit wordt ook in de Yii module

behandeld.

Voor het laatste deel van deze module heb je kennis nodig van Yii. Je hebt de Yii module dus gedaan.

Je kent verder HTML, CSS en JavaScript (liefst ES6 maar ES5 is ook goed).

Kijk nu de volgende video (60 seconden) in LinkedIn Learning en beantwoord de vragen in de Quiz.

https://www.linkedin.com/learning/learning-react-js-5/what-is-react?u=84048860 (Links to an external site.)

Note

De hele LinkedIn cursus waar deze video bij hoort, duurt één uur en is ook een goede introductie op React. Let wel op dat in deze video gebruik wordt gemaakt van *React Hook*s. Dat doen wij in deze module niet. In deze module gaan we met de tijd mee en gebruiken we OOP (classes). Hierdoor hoeven we geen gebruik te maken van relatief lastige React Hooks. Evengoed kan ik de LinkedIn cursus wel aanraden als je meer van React wilt weten.

*** Deel 1 ***

In het eerste gedeelte gaan we React installeren en gaan we kennis maken met state variabelen.

Bij een 'gewone' web pagina kan je alleen informatie opslaan met speciale technieken zoals:

1. Gebruik maken van een (SQL) database.
2. Gebruik maken van Cookies.

In React werkt dit ook, maar in React kun je ook gegevens opslaan als zogenaamde state variabelen. Deze variabelen bewaren de 'status' (state) van de applicatie.

In het eerste gedeelte gaan we leren hoe state variabelen werken.

Maar eerst zullen we React moeten installeren.

Installatie

Inleiding

Node.js is de JavaScript run-time omgeving. JavaScript draait op alle web browsers, maar met Node.js draait JavaScript ook als server.

Node.js is een verzameling van tools. Het lijkt wat dat betreft op een framework.

Installatie

Installeer Node.JS https://nodejs.org/en/download/

Maak nieuwe app en start Node.js

```
npx create-react-app <naam van de nieuwe react app>
cd <naam van de nieuwe react app>
npm start
```

Belangrijke files

package.json	alle geïnstalleerde packages
.gitignore	welke files worden door github niet naar de repo gekopieerd
public/index.html	start
source/index.js	hier begint de React applicatie - root wordt vervangen door de React app die wordt gerenderd.
source/App.js	App is de React component dat wordt geladen.

App.js is in JSX is combi HTML en JavaScript

Clean set-up

Haal nu alle bestanden weg uit de folder *public* en *src* zodat alleen de bestanden overblijven die in het onderstaande plaatje te zien zijn.

image-1626261423055.png

Verander nu de inhoud van de volgende bestanden.

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';



ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

App.js

```
function App() {
  return (
    <div className="App">
      <h1>Titel</h1>
    </div>
```

```
  );
}


export default App;
```

Start Dev Server

Via VCS voer het volgende commando uit in een *terminal* (of gebruik CMD, maar ga dan wel naar de projectfolder).

```
npm start
```

Opdracht

Als het goed is dan zie je een webpagina die er ongeveer zo uit ziet.

image-1661518073202.png

Uitleg

React start (net zoals ander web apps) met het laden van de index.html pagina.

Door het React framework wordt ook het bestand *index.js* ingelezen. Deze index.js zorgt ervoor dat het bestand *App.js* wordt gerenderd (zeg maar "uitgevoerd") en wordt geplaatst in de index pagina in de div met het id "root".

Kijk nog eens goed naar deze drie bestanden en probeer de uitleg te begrijpen. Als dat is gelukt dan kun je de opdracht maken.

Opdracht

Zorg ervoor dat de React app in plaats van "Titel" een welkomstboodschap laat zien waarin je naam is verwerkt, bijvoorbeeld:

image-1661518406825.png

Inleveren

Een schermafdruk van de browser met jouw unieke welkomstboodschap.

--

JSX en state variabelen

In REACT maken we gebruik van de 'taal' JSX. In deze les leren we wat JSX is.

Verder leren we wat in React een state variabele is en hoe we daar mee kunnen werken.

Met dit alles maken we een kleine applicatie waarmee de CSS-stijl van een kan worden aangepast door code.

Wat moet je weten?

Om deze les te kunnen volgen heb je kennis van:
Javascript (ES6), HTML, CSS en de REST API maken we in Yii, dus om deze API te maken is ook ervaring met Yii nodig.

ES6 versus ES5

React gebruikt JavaScript ES6 (ECMAScript 6). ES6 heeft op bepaalde punten een ander syntax dan ES5.

Kijk op

> https://medium.com/recraftrelic/es5-vs-es6-with-example-code-9901fa0136fc

waarin het verschil tussen ES5 en ES6 met voorbeelden wordt uitgelegd.

JSX

JSX staat voor JavaScript XML en is een mengeling van JavaScript en XML. XML is niets meer dan een formaat waarbij je telkens `<begin>..</begin>` gebruik. HTML-tags zijn eigenlijk XML tags. De XML tags in JSX lijken heel veel op HTML maar zijn hier en daar net even iets anders.

Een voorbeeld van JSX staat hieronder. Plaats deze code in de app.js die je tijdens de installatie hebt gemaakt.

```
function App() {
  return (
    <div>
      <h1 className="greeting">Hello!</h1>
      <h2>Good to see you here.</h2>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
```

```
    </div>
  );
}

export default App;
```

Dit is bijna allemaal 'gewone' HTML. Alleen *class* is geen *class,* maar heet in JSX *className*.

Verder zien we dat tussen de HTML JavaScript staat. Dit is zo omdat deze JavaScript in de HTML moet worden gezien als tekst staat deze tussen { }.

Kijk naar de volgende code die in app.js staat en probeer te voorspellen wat de code doet, voer het daarna uit en kijk of je begrijpt wat er gebeurt en waarom.

```
function App() {

  var myString = "";
  for (var i=0; i < 10; i++) {
    myString = myString + String(i) + ", ";
  }

  return (
    <div className="App">
      {myString}
    </div>
  );

}

export default App;
```

In deze functie staat een verwijzing naar een CSS class. Laten we deze toevoegen.

Maak een nieuwe file App.css in dezelfde folder als de App.js staat en zet daar het volgende in:

```
.App {
  text-align: center;
  color: red;
}
```

Voeg nu de regel: `import './App.css';` toe boven de functie App() in App.js

State

In React kun je gegevens bewaren door deze op te slaan in een speciale variabele, de state variabele. Om deze te gebruiken is het makkelijker om onze functie om te zetten in een class. Een class heeft in React twee belangrijke methods (functies); constructor() en render().

De constructor wordt eenmalig aan geroepen en hierin kun je state variabelen initialiseren.

In de render() method bepaald de output gemaakt. Je kunt dit vergelijken met de view (van MVC).

We gaan de bovenstaande functie omzetten naar het object model van React.

```
import './App.css';

import React from 'react';

class App extends React.Component {

  constructor(props) {
    super(props);
  }

  getNumbers() {
    for (var i=0; i < 10; i++) {
      myString = myString + String(i) + ", ";
    }
  }

  render() {
    return (
      <div className="App">
        {myString}
      </div>
    );
  }

}

export default App;
```

Als je deze code runt dan werkt die niet :(Dat komt omdat myString geen state variabele is. myString is alleen bekend in getNumers() en niet daarbuiten!

In de volgende stap hebben we van myString een state variabele gemaakt.

In de constructor method (deze wordt één keer aangeroepen!) wordt de state variabele gedeclareerd en daarna word de method getNumbers() aangeroepen. Met hetNumbers wordt de myString gevuld.

```
import './App.css';

import React from 'react';

class App extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      myString: ''
    }
    this.getNumbers();
  }

  getNumbers() {
    for (var i=0; i < 10; i++) {
      this.state.myString = this.state.myString + i + ", ";
    }
  }

  render() {
    return (
      <div className="App">
        {this.state.myString}
      </div>
    );
  }

}

export default App;
```

Let op, je krijgt een waarschuwing/foutmelding. Voor nu is dat even nog niet belangrijk.

Voor de gevorderen: Je kunt de *setState* functie gebruiken (zoals het hoort), maar dan moet je de int i casten naar een string. Op de manier zoals in het voorbeeld hoeft dat niet.

Opdracht

Als je het voorbeeld goed hebt uitgewerkt dan worden de nummer 0 tot en met 9 afgedrukt.

Verander de code nu zodat je de nummer 1 to en met 10 afdrukt en dat de cijfers worden gescheiden door een spatie en druk jouw naam af op de regel onder de nummers.

De output ziet er dus bijvoorbeeld als volgt uit:

image-1661519211351.png

Inleveren

Een schermafdruk van jouw output.

--

State variabelen gebruiken

We gaan de kleur van de letters aanpassen. De kleur moeten we onthouden en dat doen we in een state variabele

Dynamische CSS in JSX

We gaan nu een knop maken waarmee we de kleur van de tekst (myString) kunnen aanpassen.

We volgen de volgende stappen:

1. Knop maken die functie aanroept.
2. state variabele maken waarin de kleur wordt vastgelegd.
3. In de functie de state variabele aanpassen.
4. We maken de stijlen in de css file.
5. in de render method de style/class aanpassen; is de state variabele red dan gebruiken we een stijl waarin de tekst in rood wordt afgedrukt en is de state variabele green dan gebruiken we een stijl die de tekst groen maakt.

Stap 1, Knop maken

Allereerst maken we een button. Bij de vorige opdracht heb je jouw naam ergens laten afdrukken. Dat is de plaats waar nu de button moet komen.

```
<br />
<button onClick={() => this.changeColor() } >Change Color</button>
```

image-1661525117373.png

Als je de code nu runt en op de knop drukt, dan gebeurt er nog weinig. Dat komt omdat de method changeColor (nog) niet bestaat.

We moeten nu eerst een state variabele waarin we de kleur gaan bewaren aanmaken.

Stap 2, State variabele maken

voeg myColor toe als string aan de state variabele, let op dat je wel een komma gebruikt tussen de state variabelen.

```
this.state = {
   myString: '',
   myColor: 'red'
}
```

Nu gaan we de functie maken die vanuit de knop wordt aangeroepen maken.

Stap 3, State variabele aanpassen in functie

Een state variabele mag en kan alleen worden aangepast met een (ingebouwde functie *setState()*. Dat werkt als volgt:

```
changeColor() {
  if ( this.state.myColor === "red") {
    this.setState({ 'myColor': 'green'} );
  } else {
    this.setState({ 'myColor': 'red'} );
  }
}
```

De setState() method is ingebouwd in React en bedoeld om de waarde van state variabelen te wijzigen. De state variabelen staan in JSON formaat.

Wat deze functie doet is de variabele myColor op green zetten als die rood is, en op rood zetten als die groen is.

En werkt de knop al? Nee? Waarom niet?

Juist we moeten nog zorgen dat de kleur die wordt bewaard ook werkelijk wordt toegepast.

Eerst de CSS maken.

Stap 4, css aanpassen

In de css maken we twee extra classes.

```
.red {
  color: red;
}
.green {
  color: green;
}
```

En nu de juisten class (*red* of *green*) aan de className van de Div toevoegen.

Stap 5, class dynamisch maken

In de app.js file passen we nu de stijl aan.

De class wordt dus de waarde van de state variabele *myColor*. Deze is *green* of *red*. Deze stijlen heb je in de CSS aangemaakt in stap 4.

De class App is nu weg, laten we die ook nog even toevoegen.

```
<div className={"App "+this.state.myColor}>
```

Stel *myColor* is "red"dan staat er dus className="App red". Dit betekent dat CSS classes App en red worden toegepast.

Je kunt in de CSS file bij de classes red en green natuurlijk veel meer aanpassen. Probeer zelf maar!

Opdracht

Verander de method getNumbers().

```
getNumbers() {
  for (var i=0; i < 10; i++) {
    this.state.myString = this.state.myString + "*";
  }
}
```

Nu zie je 10 sterretjes in plaats van de nummers.

Maak nu een tweede button. Telkens als je op de button drukt laat je een extra sterretje zien. De string *myString* wordt dus telkens iets langer.

Tip: Omdat je moet onthouden hoeveel sterretjes er al staan, moet je dus een state variabele aanmaken. In de state variabele kun je dan telkens als je op de knop drukt een sterretje toevoegen.

Inleveren

De file App.js.

--

Quiz (vragen)

Hoe zou je JSX het beste omschrijven?

1. Een soort combinatie van JavaScript ES5 en JavaScript ES6.
2. Een soort combinatie van JavaScript en HTML
3. Een soort combinatie tussen JavaScript en CSS
4. Een soort combinatie tussen JavaScript, HTML en CSS.

Hoe geef je een code blok weer in JSX?

1. Tussen [en] (rechte haken)
2. Tussen (en) (ronder haken)
3. Tussen { en } (curly brackets)
4. Tussen < en > (visgraten)

Wat doet de constructor method in een class?

1. Die wordt aangeroepen als je voor de eerste de class aanroept.
2. Die wordt aangeroepen elke keer als je een class aanroept.
3. Die moet je zelf aanroepen om variabelen te initialiseren.
4. Die wordt aangeroepen als je voldoende vrij geheugen hebt en wordt gebruikt als cache (code wordt daardoor sneller).

Als je in JSX iets wilt 'onthouden' dan gebruik je wat?

1. De constructor method.
2. Een JSON array.
3. Een variabele.
4. De state variabele.

In JSX is de HTML syntax helemaal hetzelfde als 'gewone' HTML zoals we dat gewend zijn.

1. Zeker
2. Zeker maar je mag alleen hoofdletters gebruiken
3. Zeker maar er zijn kleine verschillen *class* wordt bijvoorbeeld *className*
4. Zeker alleen zet je HTML blokken tussen { en } (curly brackets).

Wat doet de render method in een JSX class?

1. Die houd zich bezig met het omzetten van code naar HTML
2. Die wordt standaard uitgevoerd als je de class aanroept en in de regel wordt daar de output gedefinieerd (soort view uit MVC).
3. Daarin wordt de HTML (opmaak) gecodeerd.
4. Daarin wordt de data uit bijvoorbeeld de database opgehaald. De data wordt dan vervolgens gestyled via de *styleName* class.

*** Deel 2 Inleiding ***

We hebben de basis gelegd in de vorige lessen. Heb je de kennis check goed gemaakt, dan kun je beginnen met het tweede gedeelte van deze module.

In het tweede gedeelte gaan we de quiz app maken.

image-1661600160072.png

Quiz App - deel 1

In deze les gaan we ook nieuwe (REACT) techniek leren. We leren wat props zijn en hoe we daarmee kunnen werken.

Verder zullen we state variabelen gebruiken, we maken gebruik van een JSON-datastructuur.

We maken grote stappen en gaan een volledige Quiz app maken in React. De app komt er zo uit te zien:

image-1628456332603.png

Testen!

We gaan deze app stapje-voor-stapje maken en we proberen elke (kleine) stap te testen. Als je programmeert dan is het handig om elk klein stapje te testen, als het dan fout gaat dan weet je goed waar je naar moet kijken. Zeker met React werkt dit heel goed want op het moment dan je jouw aanpassing bewaard zie je gelijk of je ene fout hebt gemaakt.

Soms moet je om te kunnen testen even extra (tijdelijke) code maken. Bijvoorbeeld met `console.log()`. Dit lijkt soms veel werk, maar het is de *enige* manier om een goed werkende applicatie te krijgen. Bij test-driven development maak je zelfs vaak eerst de code om te testen, dan schrijf je de code en dan test je. We zullen dit in deze les bij sommige stappen ook toepassen.

image-1628549683281.png

Lees goed en ga pas door naar de volgende stap als je begrijpt wat je hebt gedaan.

In het kort

In deze les gaan we de volgende stappen doorlopen;

1. We beginnen met een 'lege' standaard app zoals we die in de eerste les hebben gemaakt.
2. We maken een paar state variabelen (uitleg zie vorige les).
3. We maken de vragen in JSON en zetten die in een constante (dat is een variabele die niet veranderd).
4. We laten de (eerste) vraag op het scherm zien en stijlen dit een beetje.
5. We laten antwoorden zien en zorgen dat door de vragen heen kunnen lopen.

Stap 1 tot en met 4 is allemaal set-up en het echte coderen begint eigenlijk pas echt bij stap 5.

Stap 1, We beginnen met een standaard App.js

```
import './App.css';

import React from 'react';

class App extends React.Component {

  constructor(props) {
    super(props);
    this.state = {

    }
    this.getNumbers();
  }



  render() {
    return (
      <div className="app">

      </div>
    );
  }

}

export default App;
```

Stap 2, state variabelen

Tijdens het spelen van het spel moeten we bijhouden bij welke vraag we zijn en hoeveel goede antwoorden we hebben gegeven.

Maak hiervoor twee state variabelen aan; vraagNr om bij te houden bij welke vraag we zijn en score om de score bij te houden.

```
this.state = {
  vraagNr: 0,
  score: 0
}
```

Stap 3, we moeten vragen maken

We kunnen de vragen uit een database halen, maar om te beginnen zetten we de vragen gewoon even in de code als een constante. In de constante plaatsen we een JSON-data structuur met de vragen.

We gebruiken de JSON data structuur omdat we uiteindelijk de vragen uit een REST API gaan krijgen. Wat dat allemaam precies is dat wordt nog uitgelegd, maar een REST API geeft als resultaat ook een JSON-datastructuur terug. We beginnen nu dus alvast met JSON dan is de aanpassing later eenvoudiger.

Zet het volgende vlak boven de *class App* (regel 4).

```
const questions = [
    {
        questionText: 'Wat is de hoofdstad van Frankrijk?',
        answerOptions: [
            { answerText: 'New York', isCorrect: false },
            { answerText: 'London', isCorrect: false },
            { answerText: 'Paris', isCorrect: true },
            { answerText: 'Dublin', isCorrect: false },
        ],
    },
    {
        questionText: 'Wie is de oprichter van Tesla?',
        answerOptions: [
            { answerText: 'Jeff Bezos', isCorrect: false },
            { answerText: 'Elon Musk', isCorrect: true },
            { answerText: 'Bill Gates', isCorrect: false },
            { answerText: 'Tony Stark', isCorrect: false },
        ],
    },
    {
        questionText: 'Welk bedrij was eind jaren 90 bijna failliet?',
        answerOptions: [
```

```
            { answerText: 'Apple', isCorrect: true },
            { answerText: 'Intel', isCorrect: false },
            { answerText: 'Amazon', isCorrect: false },
            { answerText: 'Microsoft', isCorrect: false },
        ],
    },
    {
        questionText: 'Hoeveel Harry Potter boeken zijn er?',
        answerOptions: [
            { answerText: '1', isCorrect: false },
            { answerText: '4', isCorrect: false },
            { answerText: '6', isCorrect: false },
            { answerText: '7', isCorrect: true },
        ],
    },
    {
        questionText: 'Wat is GEEN Cryptomunt?',
        answerOptions: [
            { answerText: 'Bitcoin', isCorrect: false },
            { answerText: 'EOS', isCorrect: false },
            { answerText: 'Pancake Swap', isCorrect: false },
            { answerText: 'XOTA-XL', isCorrect: true },
        ],
    },
];
```

Stap 4a, Laat de vraag zien

De vragen staan in het JSON object en kunnen getoond worden met deze code:

```
<div>{questions[this.state.vraagNr].questionText}</div>
```

De code laat van het JSON array het eerste element (question[0]) zien. Van dat element wordt de questionText getoond.

Kijk maar eens wat er gebeurt als je deze code gebruikt:

```
<div>{questions[this.state.vraagNr].answerOptions[0].answerText}</div>
```

Kijk naar de code en naar het JSON-object en probeer te begrijpen hoe je gegevens uit het JSON-object kan laten zien.

Als je dat begrijpt, verander dan de code nu zodat je de vraag met daaronder de vier antwoorden krijgt.

image-1628538140614.png

Stap 4b, Tijd voor een beetje CSS!

Allereerst moet je weten dat de allereerste pagina die door REACT wordt geladen de file *public/index.html* is. Als je deze pagina opent dan zie je een normale HTML-layout en in de body wordt het root element aangeroepen. Deze staat in *src/index.js* en vanuit daar wordt de App.js aangeroepen.

Zoals je kunt zien in de public/index.html, wordt er een body tag gebruik. Laten we eerst de body stijlen. Voeg een App.css file toe en zorg ervoor dat de css file in de App.js file wordt geïmporteerd. In het voorbeeld van stap 1 gebeurde dat al op regel 1.

De body stijl (in App.css zetten).

```
body {
  background-color: #7cc6fe;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  font-family: "Verdana", cursive, sans-serif;
  color: #ffffff;
}
```

Nu gaan we een mooie box om de vraag heen zetten. In de app stijl, dit is de eerste div in de *App.js* file.

Maak een tweede stijl voor de app en zet deze ook in de App.css.

```
.app {
  background-color: #252d4a;
  width: 450px;
  min-height: 200px;
  height: min-content;
  border-radius: 15px;
  padding: 20px;
  box-shadow: 10px 10px 42px 0px rgba(0, 0, 0, 0.75);
}
```

Dit zorgt voor een mooie box om onze vraag heen, onze app ziet er tot nu toe zo uit:

image-1628546817642.png

Stap 5, antwoord kunnen geven

Om een antwoord te kunnen geven, moeten je op de antwoorden kunnen klikken. We doen dat door van de antwoorden buttons te maken.

```
<button>{questions[this.state.vraagNr].answerOptions[0].answerText}</button>
<button>{questions[this.state.vraagNr].answerOptions[1].answerText}</button>
<button>{questions[this.state.vraagNr].answerOptions[2].answerText}</button>
<button>{questions[this.state.vraagNr].answerOptions[3].answerText}</button>
```

Dit gaan we iets vereenvoudigen. We maken vlak na de regel *render() {* een nieuwe variabele:

```
var antwoorden=questions[this.state.vraagNr].answerOptions;
```

Omdat de variabele antwoorden nu array met vier elementen is kunne we de buttons vereenvoudigen.

```
<button>{antwoorden[0].answerText}</button>
<button>{antwoorden[1].answerText}</button>
<button>{antwoorden[2].answerText}</button>
<button>{antwoorden[3].answerText}</button>
```

Als de button wordt ingedrukt moeten we een script uitvoeren. Dit script moet bepalen of het gegeven antwoord correct is.

Dat staat in `antwoorden[0].isCorrect`, `antwoorden[1].isCorrect`, etc.

Laten we dit even controleren/testen.
Verander hiervoor **tijdelijk** bij de eerste vraag *de false in* een 0 en *de true* een 1. Deze waarden kun je namelijk makkelijker afdrukken. We passen dus onze code tijdelijk aan om te kunnen testen!

Vervang dan de buttons door de volgende code

```
<button>{antwoorden[0].answerText} {antwoorden[0].isCorrect}</button>
<button>{antwoorden[1].answerText} {antwoorden[1].isCorrect}</button>
<button>{antwoorden[2].answerText} {antwoorden[2].isCorrect}</button>
<button>{antwoorden[3].answerText} {antwoorden[3].isCorrect}</button>
```

Je ziet nu het volgende:image-1628548398163.png

De 1 achter Parijs geeft dus aan dat dat het juiste antwoord is. Natuurlijk moeten we dit weer weg halen.

Vergeet ook niet de 1-en en 0-en in de eerste vraag weer te vervangen in `true` en `false`.

We roepen nu een script aan als je op de button drukt:

```
<button onClick={() =>
this.handleAnswerOptionClick(antwoorden[0].isCorrect)}>{antwoorden[0].answerText}</button>
```

Hierboven staat één button, maak zelf de andere drie.

Als de button wordt ingedrukt dan wordt de methode (functie) this.*handelAnswerOptionClick* uitgevoerd.

Het woord *this* geeft aan dat de methode in deze Class staat.

We schrijven nu een nieuwe methode om te testen of alles goed werkt.

```
handleAnswerOptionClick(isCorrect) {
  console.log("Button clicked");
  if (isCorrect) {
    console.log("Antwoord is goed!");
  }else{
    console.log("Antwoord is fout!");
  }
}
```

Kijk naar je console log (meeste browsers CTRL-Shift-i) en test je code. Druk op de verschillende knoppen en controleer of het juiste antwoord herkend wordt.

Als alles goed is hoef je alleen nog maar naar de volgende vraag te springen als je het antwoord hebt gegeven. De huidige vraag wordt bijgehouden in de state variabele vraagNr, dus die moeten we één ophogen nadat het antwoord is gegeven.

Laten we dit in regel 7.5 (dus tussen regel 7 en 8) van de methode *handleAnswerOptionClick* doen.In de vorige les is uitgelegd dat je een state variabele alleen mag veranderen door gebruik te maken van de functie *this.setstate()*

```
this.setState( {'vraagNr': this.state.vraagNr+1} );
```

Test je code!

Oops het gaat goed tot de laatste vraag. Dat lijkt logisch want als de state variabele wordt opgehoogd en die wordt hoger dan het aantal vragen dan wordt er een vraag aangeroepen die niet

bestaat.

Om te weten wanneer we niet meer naar de volgende vraag kunnen, moeten we weten hoeveel vragen er zijn. DIt is gelijk aan de lengte van de constante *questions.* De constante *questions* is een array met 5 elementen, elk element heeft weer twee elementen en deze elementen hebben ook weer elementen. Het gaat echter om het aantal vragen en dat is het aantal elementen in de constante *questions*, 5 dus.

Laten we het even testen, in de methode (functie) *handleAnswerOptionClick* zetten we:

```
console.log("Aantal vragen: "+questions.length);
```

Voer de test uit en kijk in je console log. Zoals je kunt zien in de test is het aantal vragen 5, de vragen lopen dus van 0,1,2,3 tot en met 4. Dus in dit geval mag 3 nog worden opgehoogd en bij 4 moeten we stoppen. We zetten het ophogen van de *vraagNr* dus in een *if*.

```
if ( this.state.vraagNr < questions.length - 1) {
  this.setState( {'vraagNr': this.state.vraagNr+1} );
}
```

Test alles goed en als alles goed werkt.

We zien dus alle vragen en in de console log zien we of we het antwoord goed of fout hebben beantwoord. Telkens als we een antwoord geven springen we naar de volgende vraag en bij de laatste vraag wordt er niet meer naar de volgende vraag gesprongen. Werk dat allemaal? Top! In het volgende deel gaan we iets met punten doen.

Opdracht

Zorg dat allen goed werkt en bedenk zelf één extra vraag en voeg deze vraag toe. Bijvoorbeeld:

image-1661593315876.png

Inleveren

Als alle klaar is dan maak je een screenshot van je *gehele* scherm waarin de browser is te zien met de laatste (zelfbedachte vraag) vraag van de quiz.

Deze les is een aangepaste versie van: https://www.freecodecamp.org/news/how-to-build-a-quiz-app-using-react/

Quiz App - deel 2

We zijn bezig met het maken van een Quiz app te maken in REACT. Dit is deel twee waarin we onze App verder gaan afmaken. We leren hier ook een nieuwe techniek waarmee we in JSX door een array heen kunnen loopen.

Daarnaast gaan we met de Ternary oparatore een soort if-then-else maken in JSX.

Zorg ervoor dat je Quiz App - deel 1 goed hebt doorlopen en dat alles goed werkt.

Stap 1 - een loop in JSX met de map functie

Onze render methode in de class App ziet er tot nu toe zo uit:

```
render() {
  var antwoorden=questions[this.state.vraagNr].answerOptions;
  return (
    <div className="app">
        <div>{questions[this.state.vraagNr].questionText}</div>
        <button onClick={() =>
this.handleAnswerOptionClick(antwoorden[0].isCorrect)}>{antwoorden[0].answerText}</button>
        <button onClick={() =>
this.handleAnswerOptionClick(antwoorden[1].isCorrect)}>{antwoorden[1].answerText}
{antwoorden[1].isCorrect}</button>
        <button onClick={() =>
this.handleAnswerOptionClick(antwoorden[2].isCorrect)}>{antwoorden[2].answerText}
{antwoorden[2].isCorrect}</button>
        <button onClick={() =>
this.handleAnswerOptionClick(antwoorden[3].isCorrect)}>{antwoorden[3].answerText}
{antwoorden[3].isCorrect}</button>
    </div>
  );
}
```

Regel 6,7, 8 en 9 lijken erg veel op elkaar en kunnen in een loop worden geplaatst. De code is JSX (uitleg zie eerste les) en binnen JSX maken we een loop met de map functie. Lees hiervoor de uitleg op:

> https://www.telerik.com/blogs/beginners-guide-loops-in-react-jsx

Lees de uitleg goed en vervang de vier regels dan door de volgende code:

```
{ antwoorden.map((antwoord) => (
  <button onClick={() =>
this.handleAnswerOptionClick(antwoord.isCorrect)}>{antwoord.answerText}
{antwoord.isCorrect}</button>
))}
```

Je loopt dus door het array antwoorden heen en het element komt telkens in antwoord. Dus de eerste keer in de loop geldt:

`antwoord = antwoorden[0]`

daarna wordt

`antwoord = atwoorden[1]`

`antwoord = antwoorden[2]`

etc.

Stap 2 - Voortgang (vraagnummer)

In de state variabele *vraagNr* houden we bij bij welke vraag we zijn. Deze telt vanaf 0. We zijn dus bij vraag `vraagNr+1` en we hebben in totaal `questions.length` vragen. Dat hadden we in onze vorige les al gebruikt om te bepalen of we al bij de laatste vraag waren.

We willen op de regels afdrukken "Vraag X van Y. X is dus de state variabale vraagNr en y is de lengte van het questionsArray. We maken een aparte div zodat we de tekst straks apart kunnen stijlen.

```
<div>
  Vraag {this.state.vraagNr + 1} van {questions.length}
</div>
```

Plaats deze JSX code zodat deze regel boven aan wordt geplaatst.

Stap 3 - CSS stijlen

We maken gebruik van flex boxen; we maken één rij met twee kolommen. De hele app is één rij dus de volgende code kan bij de CSS-stijl .app worden geplaatst.

```
display: flex;
flex-direction: row;
```

```
justify-content: space-evenly;
```

We maken de twee kolommen voeg de volgende class toe in de CSS file:

```
.column {
  width: 100%;
  display: flex;
  flex-direction: column;
  flex: 1;
}
```

We maken nu de volgende structuur.

```
<div className="app">

  <div className="column">
    <span>Vraag {this.state.vraagNr + 1} van {questions.length}</span>
    <span className="question">{questions[this.state.vraagNr].questionText}</span>
  </div>

  <div className="column">
    { antwoorden.map((antwoord) => (
      <button onClick={() =>
this.handleAnswerOptionClick(antwoord.isCorrect)}>{antwoord.answerText}
{antwoord.isCorrect}</button>
    ))}
  </div>

</div>
```

Als je je code test dan zie je dat de knoppen aan de rechterkant niet mooi zijn uitgelijnd. Maak een nieuw class.

```
.uitlijnen {
  justify-content: space-between;
}
```

En zet deze class bij de tweede kolom.

```
 <div className="column uitlijnen">
```

De buttons stijlen we als volgt (plaats de code in de app.css file.

```
button {
  width: 100%;
  font-size: 16px;
  color: #ffffff;
  background-color: #252d4a;
  border-radius: 15px;
  display: flex;
  padding: 5px;
  justify-content: flex-start;
  align-items: center;
  border: 5px solid #234668;
  cursor: pointer;
}

button:hover {
  background-color: #555e7d;
}

button:focus {
  outline: none;
}
```

De app is nu al bijna af!

De enige stap is nog dat we de score moeten bijhouden en aan het eind moeten stoppen en de score laten zien.

Stap 4, Score

We hadden al een score state variabele gemaakt. We hoeven alleen nog maar een punt bij de score op te tellen als de vraag goed is beantwoord. We weten ook al wanneer een vraag goed is beantwoord want daar hebben we nog een console.log staan.

Achter de console.log waar we afdrukken dat het antwoord goed is plaatsen we dus de volgende code.

```
this.setState( {'score': this.state.score+1} );
```

Stap 5, naar het einde toe

In de method *handleAnswerOptionClick* testen we of de vraagnummer kunnen ophogen.

`if ( this.state.vraagNr < questions.length-1) {`

Als dit niet mag dan zijn bij de laatste vraag en dus bij het einde van de quiz. Als we dus een *else* maken dan komen we alleen in deze else als we de laatste vraag hebben beantwoord.

We maken een else en zetten een (nieuwe) state variabele stop op true.

```
if ( this.state.vraagNr < questions.length-1) {
  this.setState( {'vraagNr': this.state.vraagNr+1} );
} else {
  this.setState( { 'stop': true } );
}
```

Maak zelf de state variabele aan in de *construtor* en zet de waarde op `false`.

Als de state variabele stop true is dan moeten we stoppen met de quiz en moeten we geen vraag maar het eindresultaat laten zien.

We gaan de Ternary operator gebruiken in onze JSX code.

De opzet wordt als volgt:

(einde ? laat_einde_zien : laat_de_vraag_zien)

In code ziet dat er als volgt uit

```
render() {
  var antwoorden=questions[this.state.vraagNr].answerOptions;
  console.log( this.state.vraagNr, questions.length);
  return (
    <div className="app">
      { this.state.stop  ?  // hier testen we of we moeten stoppen, als we moeten stoppen laat
dan het einde zien.
        (
          <div>einde!</div>
        )
        : // dit is de else, we stoppen dus niet en laten de volgende vraag zien
        (
          <>
            <div className="column">
              <span>Vraag {this.state.vraagNr + 1} van {questions.length}</span>
```

```
              <span className="question">{questions[this.state.vraagNr].questionText}</span>
            </div>

            <div className="column uitlijnen">
              { antwoorden.map((antwoord) => (
                <button onClick={() =>
this.handleAnswerOptionClick(antwoord.isCorrect)}>{antwoord.answerText}
{antwoord.isCorrect}</button>
              ))}
            </div>
          </>
        )
      }
    </div>
  );
}
```

Kijk goed of je alles begrijpt, herken je de if - then -else? Test de code uit.

<> en </>. wtf?

Je ziet op regel 12 en 23 de tags <> en </>, dat zijn speciale JSX tags. een code blok moet namelijk altijd met één tag openen en sluiten. Als je de <> en </> weglaat dan heb je een op regel 13 en die sluit op regel 16 regel 16 moet dan het einde van het blok zijn. Je mag de <> en </> ook veranderen in een en . Het gaat erom dat de twee buitenste tags in een blok bij elkaar horen.

Stap 6, het eindresultaat

Nu laten we het einde zien. We willen natuurlijk onze score bekijken.

```
<div className='score'>
  Jouw score is {this.state.score} van de {questions.length}
</div>
```

De stijl die je kunt toepassen:

```
.score {
  display: flex;
  font-size: 24px;
  align-items: center;
```

```
}
```

Het eindscherm ziet er dan zo uit, maar je kunt natuurlijk zelf een andere stijl maken.

image-1661594536349.png

Stap 7, opruimen

Vergeet niet om de console.log statements uit jouw code te halen. Als alles klaar zijn zijn die niet meer nodig. Vanuit security is het aan te bevelen om alles wat niet nodig is weg te helen, zeker als het code is die inzicht geeft over hoe je iets hebt gecodeerd.

Inleveren

1. Lever een screendump in van jouw gehele laptop scherm met de browser waarin de uitslag van je quiz te zien is.
2. Lever de file App.js in.

--

*** Deel 3 REST API Inleiding***

In het derde gedeelte gaan we onze quiz applicatie uitbreiden met een REST API.

https://www.youtube.com/embed/IIaJYUKxN2k

Bekijk het filmpje.

Stel je voor je wilt een quiz vraag veranderen of aanpassen of toevoegen. Om dat te kunnen doen moet je de code aanpassen. De quiz vragen staan immers in de code.

We gaan een eenvoudige CRUD maken met Yii waarmee we de quiz vragen in een database kunnen opslaan. Met Yii is het maken van een standaard CRUD vrij eenvoudig. We gaan dus een database maken en met Yii maken we dan een CRUD om de database met quiz vragen te kunnen bewerken.

Dan gaan we onze Yii applicatie uitbreiden en maken we een REST API interface. Dat is in principe niet veel meer dan code die een JSON object terug geeft. Dus in plaats van HTML naar de browser te sturen worden in een REST API een JSON object naar de browser gestuurd.

In deel 2 hebben we al kennis gemaakt met een JSON object; daarin stonden immers onze quiz vragen.

Als we de Yii applicatie af hebben dan gaan we onze quiz applicatie aanpassen zodat de quiz applicatie via de REST API aan de Yii applicatie alle quiz data kan opvragen. We kunnen dan dus het JSON Object dat we in onze React code gemaakt weggooien.

Als dat eenmaal werkt dan kun je als eindgebruiker, zonder de code aan te passen, vragen bewerken, verwijderen, of toevoegen.

Deel 3 is weer onderverdeeld in twee delen:

1. maken database, Yii CRUD met REST API
2. aanpassen van onze quiz app zodat die gebruik kan maken van de database via de REST API.

Waarom REST API?

Op het web zijn veel sites als het ware met elkaar "verbonden".

Als je wat koopt bij een webwinkel kan je bijvoorbeeld via Ideal betalen. De IDEAL-website moet dan communiceren (=gegevens uitwisselen) met de webshop. Dat kan via een REST API en is veiliger dan dat de webshop rechtstreeks in de database van de bank zou kunnen kijken. Via de rest API kan je precies bepalen welke informatie met wie kan worden gedeeld.

Verder is de REST API gestandaardiseerd zodat de programmeur niet telkens een nieuwe manier hoeft te doorgronden om informatie van een andere website op te vragen.

Omdat wij met de quiz app eigenaar van de website én van de database zijn had het ook anders gekund, maar op deze manier kunnen jullie wel kennis maken met wat een REST API is en hoe die werkt.

Wij gebruiken de REST API alleen om informatie te lezen, zeg maar de 'R" van CRUD. Je kunt met een REST API alle CRUD functie maken. Dus je kunt ook een REST API maken waarmee je gegevens kan updaten, toevoegen en verwijderen

--

REST API 1

De vragen van de quiz staan als een constante in de code. In deze les gaan we een database maken en we gaan onze app aanpassen zodat je de vragen via een REST API uit de database kan halen.

Het meeste werk is om de database en REST API te maken. Dat doen we in Yii zodat we heel weinig PHP-code hoeven te gebruiken.

Database

Om data op te slaan met Yii maken we een SQL database.

Hieronder staan twee opties voor het opzetten van een database.

Bepaal voor elke optie de voor- en nadelen. Welke optie is genormaliseerd volgens de regels (van het examen)?

Optie 1

image-1628779226670.png

Optie 2

image-1628778958190.png

Omdat het in deze les niet gaat om het maken van een mooie en goede Yii app kiezen we voor optie 2. Dit zou in de praktijk **niet** de beste keuze zijn omdat je hierbij te veel gelimiteerd bent. Je kunt bijvoorbeeld geen 5 antwoorden per vraag vastleggen.

Maak een sql database "*quiz*" met één tabel, noem die "*quiz*" en maak de velden zoals beschreven in Optie 2.

image-1628885283919.png

Yii

We maken nu een standaard Yii project en maken een CRUD van onze ene tabel.

1. Maak een nieuw Yii project met de naam "*quiz*".
   `composer create-project --prefer-dist yiisoft/yii2-app-basic` quiz
   Voer dit commando uit in je CMD-window in de folder waar je het project wilt aanmaken.

Open daarna de nieuwe folder in je VSC-editor.

2. Volg de stappen uit de Yii les; vergeet niet om de:
   - web.php en de
   - db.php aan te passen (kijk naar les 1 Yii).

3. Start je development server: `php yii serve`, ga naar localhost:8080/gii en maak het model en de CRUD voor de tabel quiz.

Je hebt nu een standaard CRUD app voor één tabel. Goed, laten dan beginnen met het maken van de REST API.

Ga naar localhost:8080/quiz en gebruik de *create* om een paar vragen aan te maken. Je kunt de 5 vragen gebruiken die we hadden, maar je kunt ook andere vragen bedenken.

image-1628966220816.png

Toevoegen REST API

Het opvragen van data via een REST API is niet veel meer dan het presenteren van de gegevens in JSON. We moeten er dus voor zorgen dat onze Yii app de vragen als een JSON-bestand laten zien.

Stap 1 Security

Je kunt niet 'zomaar' vanuit een Applicatie ene andere website aanroepen en data opvragen omdat je daarmee zou kunne hacken. We moeten Yii dus vertellen dat alle API verzoeken vanaf een bepaalde server zijn toegestaan. Dat doen we door de volgende code toe te voegen aan de *QuizControllers.php*

```
return array_merge(
    parent::behaviors(),
    [
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'delete' => ['POST'],
            ],
            'class' => '\yii\filters\Cors',
            'cors' => [
            'Origin' => ['*'],
            'Access-Control-Request-Method' => ['GET', 'POST'],
```

```
            'Access-Control-Request-Headers' => ['*'],
        ],
        ],
    ]
);
}

public static function allowedDomains()
{
return [
    // '*',                        // star allows all domains
    'http://localhost:3000',
    'http://test2.example.com',
];
}
```

Regel 9 t/m 13 geven aan dat er van uit elke website een API verzoek wordt geaccepteerd als dat een GET of POST verzoek is. Wij zullen alleen GET gebruiken.

Regel 24 zorgt er voor dat we vanuit onze development omgeving toegang hebben. Regel 25 is nodig als we later de applicatie in productie zouden moeten zetten. Voor nu zou regel 25 ook weggelaten kunnen worden.

Stap 2a SQL query and JSON bestand genereren.

Voeg deze method (functie) toe in jouw *QuizController*.php

```
public function actionApiGet() {
  $sql = "select vraag, antwoord1, antwoord2, antwoord3, antwoord4, juiste_antwoord from
quiz";
  $result = Yii::$app->db->createCommand($sql)->queryAll();
  return json_encode($result);
}
```

Deze method voert een query uit en vertaald de output naar JSON. Probeer maar:

`localhost:8080/quiz/api-get`

Je ziet zoiets als dit (ik heb het even netjes onder elkaar gezet).

```
[
  {"vraag":"What is the capital of France","antwoord1":"New
York","antwoord2":"London","antwoord3":"Paris","antwoord4":"Dublin","juiste_antwoord":"3"},
  {"vraag":"Who is CEO of Tesla?","antwoord1":"Jeff Bezos","antwoord2":"Elon
Musk","antwoord3":"Bill Gates","antwoord4":"Tony Stark","juiste_antwoord":"2"}
]
```


Wij kunnen de JSON data zelf ook netter maken. In Chrome heb je een extentie waarmee je JSON data op een 'nette' manier kunt zien. De extensie heet JSON viewer.

Met de JSON viewer ziet het resultaat er dan als volgt uit:

image-1661674227804.png

Inleveren

Screen shot van je gehele scherm waarin je met je browser de API aanroept en waarin je de JSON-data ziet (zoals het voorbeeld hierboven).

--

REST API 2

Onze API is klaar maar we moeten nog iets tweaken aan het outputformaat.

Stap 1, Output formaat veranderen

Dit is wat we **nu hebben**:

```
[
  {
    "vraag": "Wat is de hoofdstad van Frankrijk?",
    "antwoord1": "New York",
    "antwoord2": "London",
    "antwoord3": "Paris",
    "antwoord4": "Dublin",
    "juiste_antwoord": "3"
  },
```

En dit is wat we **nodig** hebben in onze React app:

```
[
  {
    "questionText": "Wat is de hoofdstad van Frankrijk?",
    "answerOptions": [
      {
        "answerText": "New York",
        "isCorrect": "false"
      },
      {
        "answerText": "London",
        "isCorrect": "false"
      },
      {
        "answerText": "Paris",
        "isCorrect": "true"
      },
      {
        "answerText": "Dublin",
        "isCorrect": "false"
```

```
      }
    ]
  },
```

We moeten onze API output dus omzetten (transformeren) van het ene formaat naar het andere formaat.

We lopen in een loop vraag-voor-vraag door de vragen heen en zetten elke vraag in het juiste formaat.

Met de volgende aangepaste functie zetten we het JSON-formaat om.

```
public function actionApiGet() {

    $sql = "select vraag, antwoord1, antwoord2, antwoord3, antwoord4, juiste_antwoord from
quiz";
    $result = Yii::$app->db->createCommand($sql)->queryAll();

    $output = [];
    foreach ($result as $elem) {
        array_push( $output,
            [   'questionText' => $elem['vraag'],
                'answerOptions' =>
                [
                    ['answerText'=>$elem['antwoord1'],'isCorrect'=>( $elem['juiste_antwoord']
== 1 ? 1 : 0) ],
                    ['answerText'=>$elem['antwoord2'],'isCorrect'=>( $elem['juiste_antwoord']
== 2 ? 1 : 0) ],
                    ['answerText'=>$elem['antwoord3'],'isCorrect'=>( $elem['juiste_antwoord']
== 3 ? 1 : 0) ],
                    ['answerText'=>$elem['antwoord4'],'isCorrect'=>( $elem['juiste_antwoord']
== 4 ? 1 : 0) ],
                ]
            ] );
    }

    return json_encode($output);
}
```

Dit is een stukje lastige code. Allereerst gebruiken we de PHP-data structuren voor een array. Het array wordt dan later (in regel 20) omgezet in JSON.

Zet deze regels (tijdelijk) vlak voor de return zodat je beter kan zien wat er gebeurt.

```
echo "<pre>";
var_dump($output);
```

Dit is het array wat we in de loop (regel 7, *foreach*.....) opbouwen. Dit array wordt in regel 20 (*json_encode*) vertaald in een JSON structuur.

Stap 2, aanpassen REACT code

We passen de REACT code in twee kleine stapjes aan. We zetten eerst onze bestaande JSON in een file en laten onze code een file lezen. Als dat werkt dan hoeven we alleen nog maar de verwijzing van de file te veranderen in een verwijzing naar de REST-API (de URL dus).

We doen dit weer stap-voor-stap en zetten eerst onze bestaande JSON in een file zodat we die moeten inlezen. Het inlezen van een file is namelijk bijna hetzelfde als het inlezen van data via een REST API.

In REACT zetten we in onze public folder een nieuwe file met de naam '*data.json*'.

data.json inhoud:

```
[
   {
       "questionText": "Wat is de hoofdstad van Frankrijk?",
       "answerOptions": [
           { "answerText": "New York", "isCorrect": "false" },
           { "answerText": "London", "isCorrect": "false" },
           { "answerText": "Paris", "isCorrect": "true" },
           { "answerText": "Dublin", "isCorrect": "false" }
       ]
   },
   {
       "questionText": "Wie is de oprichter van Tesla?",
       "answerOptions": [
           { "answerText": "Jeff Bezos", "isCorrect": "false" },
           { "answerText": "Elon Musk", "isCorrect": "true" },
           { "answerText": "Bill Gates", "isCorrect": "false" },
           { "answerText": "Tony Stark", "isCorrect": "false" }
       ]
   },
   {
       "questionText": "Welk bedrij was eind jaren 90 bijna failliet?",
```

```
        "answerOptions": [
            { "answerText": "Apple", "isCorrect": "true" },
            { "answerText": "Intel", "isCorrect": "false" },
            { "answerText": "Amazon", "isCorrect": "false" },
            { "answerText": "Microsoft", "isCorrect": "false" }
        ]
    }

]
```

Test door via de url:

`localhost:3000/data.json`

de json data op te vragen.

Nu moeten we de file inlezen. Dat doen we door eerst de volgende twee methods (functies) toe te voegen aan de class *App*.

```
componentDidMount() {
  this.getData();
}

getData() {
  fetch('./data.json')
  .then(res => res.json())
  .then((data) => {
    this.setState({
      jsonLoaded: true,
      jsonData: data
    });
  })
  .catch(console.log);
}
```

De method compnentDidMount is een standaard React functie die wordt uitgevoerd als een component (class) wordt geladen.

De getdata() method hebben we zelf gemaakt en die leest de file asynchroon in. En zet de data (de vragen dus) in een state variabele; de variabale *jsonData* bevat de vragen en de *jsonLoaded* wordt op *true* gezet om te laten zien dat alle data ingeladen is.

Asynchroon betekent dat de 'code' niet wacht tot de data is ingelezen. Het inlezen gebeurt als het ware in de achtergrond.

In de render method moeten we daarom eerst wachten tot de data is ingeladen. Op de eerste regel van de render methode zetten we daarom de volgende code:

```
if ( ! this.state.jsonLoaded ) {
  return (  <div className="app">loading...</div>)
}
```

In de praktijk gaat het laden erg snel maar als het laden te lang duurt wil je niet dat je App een foutmelding geeft. We wachten dus totdat de data is ingelezen. Als dat gedaan is dan doen we:

```
var questions=this.state.jsonData;
```

Hiermee is de variabele question gelijk aan de state variabele en werkt de rest van de code weer.

We kunnen ook overal waar in de code *questions* wordt gebruikt dat vervangen door *this.state.jsonData.*

Test je code en oops.... er gaat waarschijnlijk nog iets fout.

In *handleAnswerOtionClicked* hebben we een if-statement waarmee we bepalen of we moeten stoppen. Daarin gebruiken we de constante *questions* nog. Deze moeten we vervangen in *this.state.jsonData*

Het if-statement wordt dus:

```
if ( this.state.vraagNr < this.state.jsonData.length-1) {
```

Tenslotte kunnen we bovenaan in onze code de constante *questions* weghalen.

Test nog een keer goed of alles werkt voordat je verder gaat!

De laatste stap is nu heel eenvoudig als alles goed werkt.

Stap 3, aanpassen verwijzing naar file in verwijzing naar web (Yii) web site.

Als de vorige stap goed is uitgevoerd en alles werkt dan hoeven alleen nog maar de verwijzing in de fetch naar de './json' te vervangen in de url van onze Yii app.

```
getData() {
    fetch('http://localhost:8080/quiz/api-get')
```

```
    ...
```

Je zult nu waarschijnlijk ook zien dat de data even moet laden (afhankelijk van hoe snel alles is

Nu kan je in jouw Yii app vragen aanpassen en/of toevoegen.

Inleveren

App.js

--

Installatie React packages

De volgende stappen zijn nodig voor toekomstige lessen.

Installatie React packages

React Icons

Om icoontjes in je app te laten zien.

https://react-icons.github.io/react-icons/

Open project in VSC en open terminal.

```
npm install react-icons --save
```

```
import { ICONNAME } from "react-icons/LIBNAME";
...
<iconname></iconname>
```

Zoek een icon op de website. De eerste twee letters van de naam die onder het icoontje staat is de library naam en de rest is de naam van het icoontje.

Tailwind

Tailwind is een CSS library, net zoiets als Bootstrap. Bootstrap dwingt echter een bepaalde structuur af. Over het algemeen herken je ene site dit is gemaakt met bootstrap meteen. Tailwind staat dichter bij CSS en dwingt niet een bepaalde structuur af. Tailwind staat als het ware wat dichter bij CSS, maar maakt het allemaal wel net iets gemakkelijker. Op de web site van Tailwind vind je heel veel voorbeelden en onder het menu layout vind je alle mogelijke layout opties.

Open project in VSC en open terminal.

```
npm install -D tailwindcss@npm:@tailwindcss/postcss7-compat @tailwindcss/postcss7-compat
postcss@^7 autoprefixer@^9 @tailwindcss/forms
```

Maak configuratie

```
npx tailwindcss init
```

Dit maakt de file *tailwind.config.js* in de projectfolder. Deze file passen we aan:

```
module.exports = {
  purge: ['./src/**/*.{js,jsx,ts,tsx}', './public/index.html'],
  darkMode: false, // or 'media' or 'class'
  theme: {
    extend: {},
  },
  variants: {
    extend: {},
  },
  plugins: [ require('@tailwindcss/forms')],
}
```

Deze config zorgt ervoor dat alleen de CSS die we gebruiken ook wordt toegevoegd.

Vervang de gehele inhoud van de index.css door de volgende drie regels.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Craco

React ondersteund geen post CSS. Post CSS laat JavaScript CSS styles aanpassen en wordt gebruikt door Tailwind. Hierom moeten we Craco installeren.

```
npm install @craco/craco
```

package.json (aanpassen)

```
 ...
 "scripts": {
    "start": "craco start",
    "build": "craco build",
    "test": "craco test"
  },
 ...
```

craco.config.js (nieuw maken in project folder)

```
// craco.config.js
module.exports = {
  style: {
    postcss: {
      plugins: [
        require('tailwindcss'),
        require('autoprefixer'),
      ],
    },
  },
}
```

Testen

Verander de App.js file nu in:

```
import { BiArchive } from  "react-icons/bi"

function App() {
  return (
    <div className="App container mx-auto mt-3 font-thin">
      <h1 className="text-5xl" >
        <BiArchive className="inline-block text-red-400" />Applicatie
      </h1>
    </div>
  );
}

export default App;
```

In JSX is het geen *class* maar *className*

Verder zien we allemaal class namen, deze verwijzen naar de Tailwind library.

container mx-auto	maak een container 100% breed
mt-3	top margin 0.75 rem (0.75 x standaard)
font-thin	font-weight: 100;
text-5xl	font-size: 3rem; line-height: 1;

inline-block	maak in-line block (1 regel)
text-red-400	rood 400 is normaal (meer is vetter en minder is transparant.

bron: https://tailwindcss.com/docs (gebruik search).

React Icons

https://react-icons.github.io/react-icons/

Start Dev Server

```
npm start
```

Build

```
npm run build
```

packager.json (root)

```
{
  "name": "les1",
  "version": "0.1.0",
  "private": true,
  "homepage": "http://www.softwaredeveloper.ovh/maxb/",
  ...
```

ReactNative

(Voor mobile apps)

```
npx expo init <app_name>
```