Odisee
DE CO-HOGESCHOOL

Bachelorproef

Studiegebied Industriële Wetenschappen en Technologie

Mathias Dauwe
Professionele Bachelor Elektronica - ICT
Afstudeerrichting ICT
Academiejaar 2019/2020

React Context API and Hooks compared to other Framework Hooks implementation and State Management

AppFoundry
Guldensporenpark, Gebouw I 1 bus 88
9820 Merelbeke
België

Odisee
DE CO-HOGESCHOOL

Bachelorproef

Studiegebied Industriële Wetenschappen en Technologie

Mathias Dauwe
Professionele Bachelor Elektronica - ICT
Afstudeerrichting ICT
Academiejaar 2019/2020

React Context API and Hooks compared to other Framework Hooks implementation and State Management

AppFoundry
Guldensporenpark, Gebouw I 1 bus 88
9820 Merelbeke
België

Abstract

React Context API and Hooks compared to other Framework Hooks implementation and State Management
M. Dauwe

This project investigates the new technology of React where functional programming is made possible. Next these technologies are compared to other framework implementations. The comparison will be about the Hooks implementation and the state management of other frameworks.

In the first chapter of the report the internship company is presented. Next the description of this report is summarized. After this chapter it is possible to consult the action plan, which is a timeline with an overview of the tasks weekly assigned. The following chapter describes the preliminary study where the global functionality of React is explained. Also the hooks and Context API are theoretically and by examples examined. Finally the different frameworks which will be used in the comparison are summarized. Then a practical working-out gives the actually comparison between the different frameworks executed by hook. The last chapter contains the benchmarks of the different frameworks used in the practical working-out.

Keywords: React, React Hooks, Context API, framework, State Management

Voorwoord

In de opleiding van Professionele Bachelor in de Elektronica – ICT aan Odisee is het maken van een rapport voor de bachelorproef een opdracht ter eindevaluatie van de opleiding. Deze opdracht bevat de uiteindelijke weergave van een systematische zoektocht, gestoeld op de toepassing van wetenschappelijke kennis, creativiteit en praktijkkennis. Hierbij staat zelfstandigheid en bijsturing voorop.

Het thema voor deze proef levert een bijdrage tot een voor het werkveld relevant thema. Vanuit AppFoundry is het onderwerp aangereikt aangezien deze nieuwe technologie dagdagelijks gebruikt wordt. De link met de eigen interesse met React om frontend applicaties te schrijven heeft de keuze voor dit onderwerp enkel bevorderd.

Graag een dankwoord aan de interne en externe begeleiders voor de bachelorproef en stage. De goede raad en tips hebben ervoor gezorgd dat dit project tot een goed einde werd gebracht.
Ook had ik graag een blijk van appreciatie gegeven aan AppFoundry voor de kans om mijn stage binnen dit bedrijf te mogen uitvoeren.
Hiernaast wil ik ook mijn waardering geven aan het volledige docentenkorps van de opleiding Elektronica – ICT voor de excellente begeleiding en de boeiende leerstof.
Tot slot wens ik mijn ouders nog te bedanken voor de kans die mij werd gegeven om verder te studeren.

Inhoud

Figurenlijst

Tabellenlijst

Codefragmentenlijst

Inleiding

De projectopdracht bestaat erin een studie te voeren met als onderwerp React Context API en Hooks vergeleken met de implementatie van hooks en state management bij andere frameworks. Hierbij wordt een vergelijkende studie gevoerd tussen drie verschillende frameworks. Daarbij wordt enerzijds gekeken naar de implementatie van hooks binnen andere frameworks, en anderzijds de state management.

Het project wordt opgesplitst in twee delen. Eerst is het de bedoeling een uitvoerige studie te voeren in verband met React Hooks en Context API zodat de werking van deze concepten duidelijk is. Het is de bedoeling de achtergrond van de ontwikkeling van hooks en context te achterhalen. Nadien volgt een onderzoek over de werking van de Context API, en vervolgens worden de hooks afzonderlijk bestudeerd. Beide gebeuren zowel praktisch, als theoretisch. Nadien is het de bedoeling binnen de praktische implementatie deze concepten te vergelijken met implementaties binnen andere frameworks. Op die manier is het mogelijk een studie te voeren tussen de verschillende technieken en hieruit ook een gepast besluit te trekken. De vergelijking wordt beperkt tot de implementatie van de hooks en de Context API.

1. Voorstelling van het stagebedrijf

Het bedrijf waarvoor de opdracht wordt uitgevoerd is AppFoundry. AppFoundry is ontstaan in de zomer van 2014 en beschikt momenteel over een personeelsbestand van 21 werknemers. Het bedrijf zet zich voornamelijk in voor de ontwikkeling van web- en mobiele applicaties. [1]

AppFoundry is een consultancy bedrijf dat beschikt over drie satelietkantoren. Naast het hoofdkantoor in Kontich, heeft AppFoundry nog twee satelietkantoren in Hasselt en Merelbeke. Aangezien AppFoundry deel uitmaakt van de Xplore Group, bevinden zij zich in drie van de negen kantoren van hen. Xplore Group is een dochteronderneming van De Cronos Groep. De Cronos Groep helpt mensen hun ondernemingsidee tot stand te brengen. Op die manier worden mensen begeleid bij hun onderneming. [2]

Op de palmares van AppFoundry zijn reeds een aantal interessante projecten terug te vinden. Zo is de applicatie voor het Sportpaleis, de Argenta bank, de VRT Sporza Voetbal, … hierin terug te vinden. Niet enkel de mobiele applicaties, maar ook applicaties voor de smartwatches worden hier ontwikkeld. Onder andere de Belfius app op de horloge is een project van AppFoundry. [3]

2. Stageopdracht

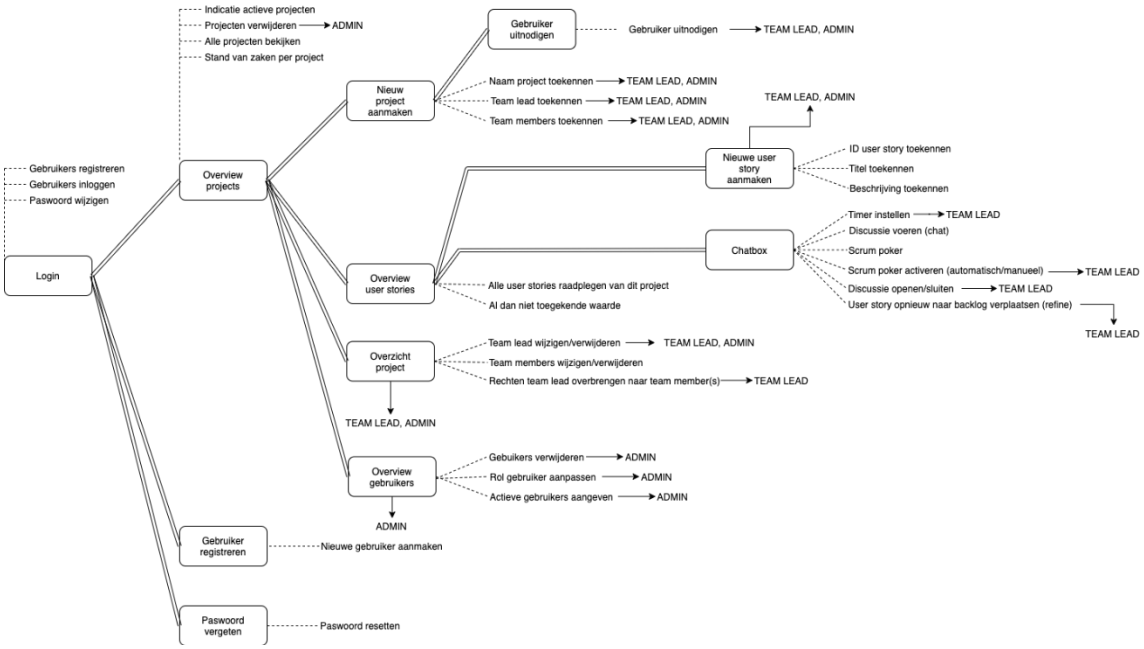
De stageopdracht is voorgesteld vanuit het standpunt van AppFoundry en krijgt de naam Cahoot. Cahoot is een programma dat als doelstelling heeft scrum poker mogelijk te maken op afstand en volledig asynchroon. Dit houdt in dat de verschillende deelnemers niet op dezelfde fysieke plaats aanwezig moeten zijn en dat verschillende user stories gelijktijdig uitgevoerd kunnen worden.

Het is de bedoeling om het volledige proces te doorlopen van analyse tot eindproduct met behulp van de scrum techniek.

2.1. Analyse

Bij de start van de stage werd de opdracht toegelicht. Vanaf dit punt was het de bedoeling in verschillende stappen tot een duidelijk beeld te komen. Zo zijn de volgende stappen uitgevoerd in de beschreven volgorde:

- Persona's maken
- User stories opstellen
- Flow diagram aanmaken (Figuur 1)
- Technische benodigdheden onderzoeken
- Grafische voorstelling maken
- Belangrijkste user journeys maken



Figuur 1: Flow diagram Cahoot

2.2. Proces

Na de analyse startte de scrum methodologie en werden wekelijkse sprints gepland. Dit biedt ook de mogelijkheid bij een foute interpretatie een bijsturing te voorzien.

3. Omschrijving van de bachelorproef

AppFoundry heeft de aanzet gegeven tot het onderwerp van de bachelorproef, namelijk de studie van React Hooks en Contexts. Om tot een voldoende diepgaand onderwerp te komen is de vergelijking met andere frameworks aan het onderwerp toegevoegd.

Dit houdt in dat React Hooks en Contexts theoretisch, maar ook praktisch worden bestudeerd binnen het React framework. Hierbij wordt de performance getest tussen de klasse- en functionele componenten. Ook de beperkingen worden bekeken en besproken samen met de voor- en nadelen.

Nadien wordt de vergelijking gemaakt met andere frameworks hoe zij deze methodologie aanpakken. Hierbij wordt de performance tussen de verschillende frameworks getest, maar evengoed de gebruiksvriendelijkheid wordt bekeken.

Tot slot wordt een conclusie gevormd over de resultaten die uit de vergelijking gehaald worden en de benchmarks van de verschillende frameworks, waarna deze uitvoerig besproken worden.

4. Actieplan

A: Stageopdracht
B: bachelorproef

	Odisee Studiegebied IWT Opleiding Electronica-ICT 2019-2020	Technologiecampus Gent Gebr. Desmetstraat 1 9000 GENT Tel.: (09) 265 86 10 Fax: (09) 225 62 69

ACTIEPLAN (vereisten op pg. 2)

Student(e): Mathias Dauwe — Groep: 3ICT

Stageplaats: AppFoundry

Stageleid(st)er (interne promotor): Yves Blancquaert

Stagementor (externe promotor): Jens Goeman

Stap	Inhoud	Streef-datum	Werkelijke datum	Opvolging
1A 1B	Analyse Blog opzetten, actieplan maken en titel bachelorproef bedenken	Einde Week 1	A: Week 1 B: Week 1	
2A 2B	Kennismaking met de verschillende technieken Vastleggen titel bachelorproef en richtlijnen doornemen	Einde Week 2	A: Week 2 B: Week 2	
3A 3B	Login functionaliteit voorzien Start opmaak bachelorproef voor eerste indiening	Einde Week 3	A: Week 4 B: Week 2/3	
4A 4B	Project overview screen voorzien Verder werken opmaak bachelorproef	Einde Week 4	A: Week 6 B: Week 4	
5A 5B	Implementatie "UserStoryScreen" en "CreateUserStoryScreen" Algemene info in bachelorproef	Einde Week 5	A: Week 7, 8 B: Week 5	
6A 6B	Chatbox voorzien Algemene info in bachelorproef	Einde Week 6	A: Week 9 B: Week 6	
7A 7B	Chatbox voorzien Maken van de vergelijkende programma's en documentatie	Einde Week 7	A: Week 9 B: Vrije week	
8A 8B	Implementatie "CreateProjectScreen" en "InviteUsersScreen" Maken van de vergelijkende programma's en documentatie	Einde Week 8	A: Week 7 B: Week 10	
9A 9B	Implementatie "ProjectInformationScreen" Documentatie uitgevoerde tests en gemaakte programma's	Einde Week 9	A: ... B: Week 11	
10A 10B	Implementatie "UsersScreen" Extra praktische oefeningen maken voor bachelorproef	Einde Week 10	A: ... B: Week 11	

Figuur 2: Actieplan pagina één

11A 11B	Bugfix Extra oefeningen documentateren	Einde Week 11	A: Niet B: Week 12
12A 12B	Refinement Nalezen bachelorproef	Einde Week 12	A: Niet B: Na stage
13A 13B	Refinement Extra aanvullen bachelorproef en dubbele verificatie	Einde Week 13	A: Niet B: Na stage
14B	Bachelorproef verder afwerken tot een volwaardige eindgeheel	Vervolg	B: Na stage

Figuur 3: Actieplan deel twee

5. Voorstudie

Alvorens een studie te starten is een voorstudie van groot belang. In onderstaande hoofdstukken worden de globale aspecten besproken die gebruikt worden doorheen de studie. Hierbij is het dus de bedoeling de theoretische achtergrond voor te stellen, zodat verdere diepgang mogelijk is.

5.1. React

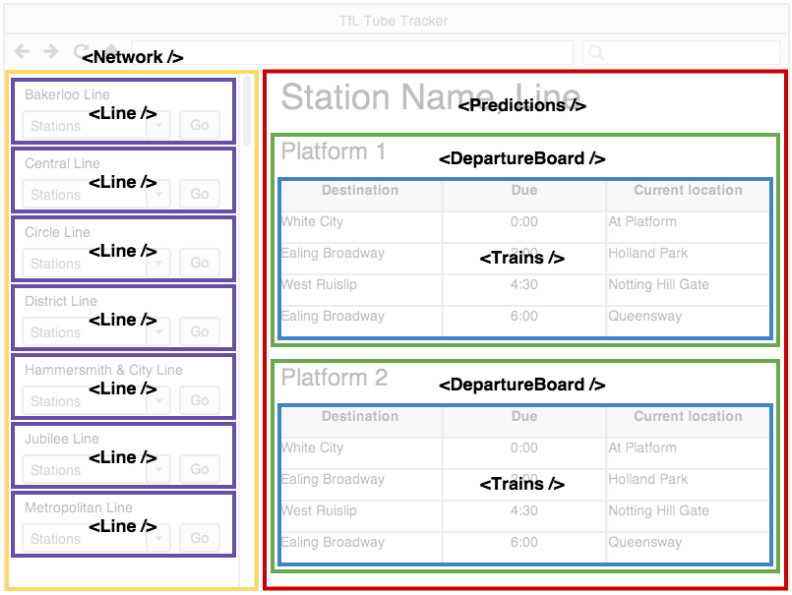
React is geen frontend JavaScript framework, wat door het merendeel wel wordt beschouwd, maar een JavaScript-bibliotheek. Deze wordt gebruikt om gebruikersinterfaces op te bouwen en grote projecten te maken met het inzicht om deze bijgevolg beter beheersbaar te maken. [4]

Deze JavaScript-bibliotheek is ontwikkeld door Facebook en wordt hierdoor ook onderhouden. Aangezien de codebase van Facebook miljoenen lijnen code bevat was het na bepaalde tijd niet meer onderhoudbaar. Facebook was genoodzaakt om veranderingen door te voeren, en deze eerste veranderingen namen plaats in 2011. Op dat moment werd xhp geïntroduceerd in de PHP stack. Deze syntax is later voorgesteld in de syntax van React. Dit betekende meteen ook de eerste tekenen van React. Het ontstaan van React heeft twee jaar in beslag genomen waarna ze besloten hebben React opensource te maken. Sindsdien is React blijven opdraven met nieuwe updates en features waardoor het kon uitgroeien tot één van de populairste ter wereld. Het vertrouwen is mede te danken aan het feit dat achter React een sterk bedrijf gevestigd is, mede door de eenvoudige structuur die gehanteerd wordt. [5]

5.1.1. Toelichting

Om het verder verslag door te nemen is het aangeraden een voorkennis te hebben van React. Indien dit niet het geval is, is het aangeraden onderstaande toelichting door te nemen.

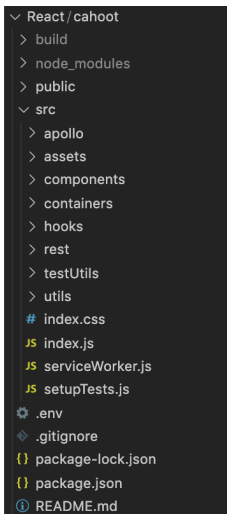
Dit framework is gebaseerd op component ontwikkeling. Oorspronkelijk wordt per HTML-pagina, of beter bekend als HyperText Markup Language pagina, een HTML-bestand voorzien. Dit brengt veel dubbele code met zich mee, zoals bijvoorbeeld de header die op elke pagina aanwezig is. Oorspronkelijk was het de bedoeling deze op elke pagina aan te maken. Met de nieuwe filosofie van component ontwikkeling wordt de UI (User Interface) opgedeeld in verschillende secties. Een sectie kan gezien worden als één geheel zoals een header, footer, knop, card, …



Figuur 4: Opbouw pagina in componenten [6]

In bovenstaande afbeelding is de opbouw van een pagina terug te vinden. Opvallend is dat de pagina in twee delen verdeeld is, namelijk Network en Predictions. Deze twee delen zijn beide componenten. Wanneer het vervolgens noodzakelijk is op een andere pagina het Network component opnieuw te voorzien, is het van belang enkel het Network component te importeren zonder de code voor dit component opnieuw te schrijven. Op de plaats van het Predictions component is het mogelijk een nieuw component of invulling te voorzien. Binnen het Network component zijn zeven Line componenten terug te vinden. Het voordeel bij deze toepassing is dat de code voor het Line component slechts eenmalig voorzien is binnen het programma. Nadien is het enkel noodzakelijk zeven maal het component te importeren. Deze denkwijze gaat vervolgens verder in het Predictions component, waarin twee DepartureBoard componenten terug te vinden zijn. Ook hier is het mogelijk slechts eenmalig de code voor het DepartureBoard component te voorzien. Nadien moet deze enkel geïmporteerd worden in het Predictions component. Deze denkwijze gaat verder voor het Trains component. (Figuur 4)
Om tot slot het Line component binnen het Network component te integreren, wordt gebruik gemaakt van een Custom HTML-tag. Deze tag gaat voor het Line component er als volgt uitzien: “<Line />”. Om bovenstaand voorbeeld te verkrijgen wordt deze tag zeven maal aangeroepen binnen het Network component.

Na de verduidelijking in verband met de componenten is het noodzakelijk de structuur van een React applicatie te begrijpen.



Figuur 5: Folderstructuur React applicatie

Bovenstaande afbeelding geeft verduidelijking in verband met het verloop van de applicatie. Wanneer de applicatie wordt opgestart, wordt index.js geopend. In dit bestand wordt voornamelijk de globale configuratie van het project voorzien aangezien dit de root is. Vanuit dit bestand wordt verwezen naar de containers-folder waarin reeds ook een index.js bestand voorzien is. In dit bestand bevindt zich de navigatie van de applicatie, waarbij elke link verwijst naar een specifieke container. Een container is ook een component, maar deze stelt de volledige pagina voor, waarin andere componenten geïmporteerd worden zoals een knop, een lijst. In dit geval zou Figuur 4 nog omringd zijn met een andere component, een container.
Binnen deze containers wordt bijgevolg de algemene logica van deze pagina geplaatst. Zoals vermeld wordt binnen een container andere componenten geïmporteerd die zich bevinden in de components-folder.
De assets-folder bevat de CSS-bestanden, Cascading Style Sheets, voor de containers en de componenten.
Hooks dat later wordt toegelicht en die afgezonderd kunnen worden, worden bijgehouden in de hooks-folder. Dit gaat voornamelijk over de Reducer Hooks en Custom Hooks, aangezien de andere hooks binnen het component gebruikt moeten worden.
De overige folders die terug te vinden zijn worden voornamelijk gebruikt voor herbruikbare functies. De rest-folder bevat bestanden die nodig zijn voor de interactie met de backend, en de utils-folder bezit functies die herbruikbaar zijn voor de logica eenvoudiger te maken. (Figuur 5)

Oorspronkelijk was de filosofie van React om gebruik te maken van klasse componenten, maar momenteel wordt deze verandert naar het gebruik van functionele componenten. Dit heeft te maken met de opbouw van een component. Aangezien bepaalde functionaliteiten van een klasse component niet mogelijk waren in functionele componenten, zijn hooks op de markt gebracht om deze tekortkomingen te verhelpen. [7]

5.2. React Hooks

Op 6 februari 2019 heeft React versie 16.8 op de markt gebracht. Dit is een versie waarbij een belangrijk probleem onder de loep werd genomen en opgelost. Vanaf dit moment is het

mogelijk om lokale variabelen binnen een component en andere React features, zoals bijvoorbeeld lifecycle methoden, te hanteren zonder een klasse te hoeven schrijven. [8]

De reden voor de ontwikkeling van React Hooks is te danken aan volgende problemen:

- Wrapper Hell;
- Grote componenten;
- Verwarring bij het gebruik van klassen;

Er zijn twee manieren voor het delen van logica, namelijk higher-order componenten en render props. Een higher-order component is een functie waaraan een component wordt meegegeven, deze manipuleert en vervolgens terugstuurt. Render props zijn variabelen die aan een Custom HTML-tag meegegeven worden, waardoor het mogelijk is deze variabelen te gebruiken in dit component. Op die manier worden variabelen van een parent component naar een child component doorgegeven.
Bij grote applicaties is het probleem dat er gesproken wordt over een groot aantal geneste componenten, wat de benaming “Wrapper Hell” krijgt. (Figuur 1) [9] [10]

Figure 1: Voorbeeld Wrapper Hell [11]

Alvorens dieper in te gaan op React Hooks is het belangrijk een onderscheid te maken tussen functionele en klasse componenten. Het klasse component erft telkens over van de Component klasse. Hierdoor is het mogelijk toegang te krijgen tot de state en lifecycle methoden. Deze state en lifecycle methoden zijn onmisbaar bij het maken van een project in React.
State wordt gebruikt om de waarde van de variabelen bij te houden en deze op verschillende plaatsen in de klasse te manipuleren en voor te stellen.
Daarnaast zijn ook lifecycle methoden belangrijk bij het maken van een klasse. Dit zijn methodes die op verschillende fases van het component uitgevoerd worden. Zo bestaat een methode wanneer het component aangemaakt wordt en geïnjecteerd wordt in de DOM, Document Object Model. Dit wordt ook wel “mounting” genoemd. Hiernaast bestaan ook

lifecycle methoden voor het updaten van het component en het verwijderen van het component uit de DOM, genaamd “unmounten”. [12]

Naast de klasse component, bezit React zoals eerder vermeld ook functionele componenten. Functionele componenten zijn zuivere JavaScript functies die niet in het bezit zijn van een render-functie zoals een klasse component. Dit omwille van het feit dat het niet mogelijk is een functie in een functie te schrijven.

In de versies voordien was een klasse component niet weg te denken wanneer een applicatie geschreven werd, aangezien de state en de lifecycle methoden noodzakelijk zijn. Eén van de onderzoeken die gebeuren is het testen of een applicatie geschreven kan worden zonder het gebruik van klasse componenten. Dit gebeurt aan de hand van de voorziene React Hooks. De voornaamste reden voor deze test is de reductie aan code om een functionaliteit te bekomen.

Een ander voordeel dat React Hooks met zich meebrengt is de mogelijkheid tot het hergebruiken van stateful logic. Dit is logica dat wordt gebruikt in verschillende componenten in combinatie met de state. Op die manier wordt herhaling van code vermeden en onderhoud bevorderd.
Het volgende voordeel betreft de opsplitsing van de code. Voordien werd alles in de lifecyle methoden gestoken in een klasse component, wat na verloop van tijd niet onderhoudbaar was. Vanaf heden is het mogelijk het klasse component op te splitsen in afzonderlijke delen, op basis van hun functionaliteit. Dit is mogelijk aangezien vanaf heden de lifecycle methoden te implementeren zijn in functionele componenten. [13]

Een belangrijk aandachtspunt bij het gebruik van React Hooks is dat deze werken op volgorde. Wanneer bijgevolg één van de opeenvolgende hooks niet wordt uitgevoerd omwille van een if-statement, gaan de andere een foutief resultaat geven. Dit omwille van het feit dat elke hook één plaats wordt opgeschoven. [14]

Ondertussen zijn reeds verschillende React Hooks ontwikkeld:

- useState
- useEffect
- custom hooks
- useContext
- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

5.2.1. useState

Om een klasse component te kunnen elimineren is de React Hook “useState” noodzakelijk. Via deze weg is het mogelijk de state bij te houden, te veranderen en voor te stellen.

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

function StateContainer() {
  const [fruit, setFruit] = useState('Apple');

  return (
    <Fragment>
      <p>{ fruit }</p>
      <button onClick={() => setFruit('Orange')}>
        Change fruit to orange
      </button>
    </Fragment>
  );
}

export default StateContainer;
```

Codefragment 1: Voorbeeld useState

In Codefragment 1 is het mogelijk te achterhalen hoe de implementatie, initialisatie en voorstelling gebeurt aan de hand van de React Hook useState. Bij de declaratie is te zien dat “array destructuring” wordt toegepast. Dit is een JavaScript expressie om de data van een array toe te kennen aan verschillende variabelen. [15]

5.2.2. useEffect

Een noodzaak om een functioneel component dezelfde mogelijkheden te laten bieden als een klasse component is het bezitten van lifecycle methoden. Dit probleem is verholpen door de ontwikkeling van de React Hook useEffect.
Volgende lifecyle methoden worden gebruikt:

- componentDidMount(), voor de weergave van het component;
- componentDidUpdate(), na update van de DOM;
- componentWillUnmount(), na de weergave van het component;

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

function EffectContainer() {
  const[fruit, setFruit] = useState('Apple');

  // Replacement for componentDidMount and componentDidUpdate
  useEffect(() => {
    document.title = `The fruit for this moment is ${ fruit }!`;
  });

  return (
    <Fragment>
      <p>{ fruit }</p>
      <button onClick={() => setFruit('Orange')}>
        Change fruit to orange
      </button>
    </Fragment>
  );
}

export default EffectContainer;
```

Codefragment 2: Voorbeeld useEffect

Dit is uitsluitend een basisvoorbeeld voor het gebruik van de React Hook useEffect. In dit voorbeeld kan het worden aanzien als het aanspreken van de lifecycles componentDidMount() en componentDidUpdate() van een klasse component. Aangezien deze in een functioneel component niet bestaan, wordt gesproken over het uitvoeren van

useEffect bij elke (re-)render. Dit is een geschikte methode voor het binnenhalen van gegevens alvorens de UI wordt opgebouwd.
De hook useEffect is een verzameling van alle lifecycle methoden gecombineerd in één functie. Hierdoor is de reductie van code aanzienlijk ten opzichte van het schrijven van 3 functies waarbij code wordt herhaald. Wanneer de componentWillUnmount() aangeroepen wordt is het de bedoeling binnen de useEffect een return functie te voorzien. In deze functie bevindt zich de logica die uitgevoerd wordt voor het component unmount. Dit is het moment wanneer het component wordt verwijderd van de DOM.

Een voordeel dat useEffect met zich meebrengt is dat vaak voorkomende bugs vermeden worden. Een vaak voorkomende bug is het uitvoeren van de code bij een update. Bij een klasse component is het de bedoeling een aparte functie te voorzien waarbij gelet moet worden om eerst code uit te voeren voor een unmount alvorens de code voor de mount. Dit brengt een extra aandachtspunt mee, wat bij useEffect niet het geval is. Deze denkwijze is reeds ingebakken in de implementatie van de hook. [16]

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

function EffectContainer() {
  const[fruit, setFruit] = useState('Apple');
  const[count, setCount] = useState(0);

  // Replacement for componentDidMount and componentDidUpdate
  useEffect(() => {
    console.log('I edit the title of the page.');
    document.title = `The fruit for this moment is ${ fruit }!`;
  }, [fruit]);

  return (
    <Fragment>
      <section>
        <p>{ fruit }</p>
        <button onClick={() => setFruit('Orange')}>
          Change fruit to orange
        </button>
      </section>
      <section>
        <p>{ count }</p>
        <button onClick={() => setCount( count + 1 )}>
          Change the counter
        </button>
      </section>
    </Fragment>
  );
}

export default EffectContainer;
```

Codefragment 3: Voorbeeld useEffect re-render

In Codefragment 3 wordt aangegeven hoe ongewenste re-render van een component wordt tegengegaan. Oorspronkelijk wordt een re-render uitgevoerd bij elke update van de state of de props. In deze denkwijze kan besloten worden dat bij elke klik op een knop, zowel fruit als teller, een re-render veroorzaakt wordt. Dit is niet het geval omwille van de array die wordt meegegeven aan de useEffect-functie. Dit geeft aan dat enkel een re-render uitgevoerd wordt wanneer de state wijzigt tussen de vorige en huidige waarde van het fruit. De teller heeft geen invloed hierop.

5.2.3. Custom Hooks

Naast de bekende hooks is het mogelijk om willekeurige hooks te maken. Dit biedt als voordeel dat logica kan afgescheiden en hergebruikt worden in verschillende functionele componenten. Dit vermindert de hoeveelheid dubbele code, wat een meerwaarde biedt aan de onderhoudbaarheid.

Het volgende voordeel is dat in custom hooks ook andere hooks gebruikt kunnen worden. Op die manier is het mogelijk de state bij te houden en te werken met lifecycle methoden. [17]

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

function useSpeed(index, period) {
  const [miles, setMiles] = useState(index);

  useEffect(() => {
    let interval = setInterval(() => {
      setMiles(miles + 1);
    }, period);
    return function cleanup() {
      clearInterval(interval);
    }
  });

  if(miles <= 100) {
    return miles;
  }
  return 100;
}

export default useSpeed;
```

Codefragment 4: Voorbeeld custom hook

In Codefragment 4 is een custom hook dat twee paramaters ontvangt, index en period. Hierin wordt useEffect gebruikt om bij mount een interval te activeren en bij unmount te deactiveren. Vervolgens zijn twee componenten aangemaakt Porsche en Seat die beide de useSpeed gebruiken. Op die manier kan de logica van de custom hook in beide componenten gebruikt worden.

5.2.4. useContext

De uitgebreide uitleg in verband met React Context staat uitgelegd in 5.3 React Context. De werking is inmiddels volledig hetzelfde, enkel de manier voor het opvragen van het context-object is nu eenvoudiger. Dit geeft de waarde weer die is vastgelegd bij de dichtstbijzijnde provider.
Wanneer de context wordt aangepast, dan wordt een automatische re-render gegenereerd.

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

// Context
import { MyContext } from '../contexts/MyProvider';

// assets/img
import Awake from '../assets/img/Awake.jpg';
import Sleeping from '../assets/img/Sleep.jpg';
import Doubt from '../assets/img/Doubt.jpg';

// assets/css
import '../assets/css/StatusPerson.scss';

function StatusPerson() {
  const context = useContext(MyContext);

  let image;
  if (context.state === 'Day') {
    image = <img src={ Awake } alt="Awake" />
  }
  else if (context.state === 'Night') {
    image = <img src={ Sleeping } alt="Sleeping" />
  }
  else {
    image = <img src={ Doubt } alt="Doubt" />
  }
  return (
    <div onClick={context.changePeriod} className="StatusPerson__Container">
      { image }
    </div>
  );
}

export default StatusPerson;
```

Codefragment 5: Voorbeeld useContext

Codefragment 5 geeft aan hoe useContext moet aangesproken worden. In dit voorbeeld wordt aangegeven hoe de context aangesproken kan worden en gemanipuleerd. Dit is een vervanging voor het functioneel component: Landscape. (Codefragment 15)
Het is inmiddels wel noodzakelijk de voorgaande stappen beschreven in 5.3 te doorlopen. (Codefragment 13 en Codefragment 14) [18]

5.2.5. useReducer

Deze React Hook is te vergelijken met useState. Het is mogelijk om variabelen te bewaren, enkel is het nu mogelijk eventuele logica te voorzien per bewerking. Via deze manier worden verschillende voorgedefinieerde niveaus voorzien. Met behulp van een dispatch-functie is het mogelijk de vooropgestelde logica op de variabelen uit te voeren. Na het uitvoeren is het mogelijk de aangepaste variabele rechtstreeks aan te spreken. Dit wordt veroorzaakt aangezien het component opnieuw wordt gerenderd na het uitvoeren van een dispatch.

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

function reducer(state, action) {
  if (action.type === 'heavier' && state.weight !== 10) {
    return { weight: state.weight + 1 };
  }
  else if (action.type === 'reduce' && state.weight !== 0) {
    return { weight: state.weight - 1 };
  }
  return state;
}

function ReducerContainer() {
  const [state, dispatch] = useReducer(reducer, { weight: 1 });

  return (
    <div>
      <p>How much kilo can you carry?</p>
      <p>{ state.weight } lbs</p>
      <button onClick={() => dispatch({ type: 'heavier' })}>Heavier!</button>
      <button onClick={() => dispatch({ type: 'reduce' })}>Please reduce...</button>
    </div>
  );
}

export default ReducerContainer;
```

Codefragment 6: Voorbeeld useReducer

In bovenstaande afbeelding is een voorbeeld te zien van useReducer. Hier is de initialisatie en het gebruik ervan te raadplegen. Met behulp van de dispatch-functie in het button-element is het mogelijk de reducer uit te voeren. Op die manier wordt de state gemanipuleerd en vervolgens teruggestuurd naar de UI. Wanneer de variabele niet verandert van waarde na uitvoering van de reducer, dan wordt het component niet opnieuw gerenderd. (Codefragment 6)
Het is mogelijk aan de useReducer-functie een derde parameter toe te kennen. Dit is een functie die het mogelijk maakt een initiële waarde mee te geven. Deze functie is handig wanneer een reset wordt voorzien. [19]

5.2.6. useCallback

In eerste instantie kan verwarring ontstaan over de werking omwille van de naam. Dit ondanks het feit dat een callback-functie toegepast wordt bij het uitvoeren van een asynchrone taak. Dit is bijgevolg de functie die in actie treedt wanneer de asynchrone taak beëindigd wordt.
In deze versie wordt er een gecachte versie van de functie bijgehouden wat ervoor zorgt dat overbodige rendering vermeden wordt. Deze Callback Hook is het best uit te leggen aan de hand van een voorbeeld.

In onderstaande afbeelding zijn twee functies te zien, een functie met de Callback Hook en een functie zonder. Het grote verschil tussen deze functies bevindt zich achter de schermen. Dit houdt in dat bij elke rendering een nieuwe instantie wordt aangemaakt in de constante “increment”. Terwijl er pas een nieuwe instantie wordt aangemaakt in de constante “incrementCallback” wanneer een wijziging optreedt in de variabele “callbackCount”. Anders wordt een gecachte weergave van de functie in de constante “incrementCallback” opgeslagen. (Codefragment 7)

```
const incrementCallback = useCallback(() => {
  setCallbackCount(callbackCount + 1);
}, [callbackCount]);

const increment = () => {
  setCount(count + 1);
};
```

Codefragment 7: Voorbeeld useCallback

Deze optie wordt gebruikt wanneer de performantie een belangrijke rol speelt, of problemen met performantie aanwezig zijn. [20] [21] [22]

Indien de werking van de Callback Hook nog niet duidelijk is, kan meer informatie achterhaald worden in het hoofdstuk: 5.2.7 useMemo. (Codefragment 8)

5.2.7. useMemo

In plaats van een gememoriseerde callback, zoals bij de Callback Hook, wordt bij de Memo Hook gebruik gemaakt van een gememoriseerde waarde.

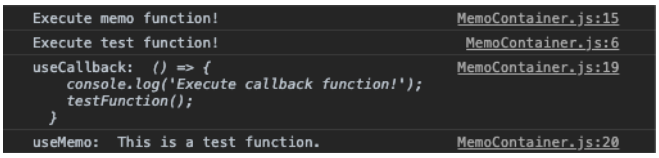
De hook ontvangt als eerste parameter een functie en als tweede een array van afhankelijkheden, of beter bekend als dependencies. Wanneer minstens één van de dependencies verandert, geeft de hook een nieuwe waarde. De opbouw is bij deze sterk gelijk aan de Callback Hook, maar het verschil zit in de teruggestuurde waarde van de hook. In de Callback Hook wordt de functie teruggestuurd, en bij de Memo Hook wordt de waarde dat uit de functie komt teruggestuurd.

Aan de hand van onderstaand voorbeeld wordt de werking van de Memo Hook en de Callback Hook duidelijk.

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

function MemoContainer() {
  const [count, setCount] = useState(0);
  const testFunction = () => {
    console.log('Execute test function!');
    return 'This is a test function.';
  };

  const callbackFunction = useCallback(() => {
    console.log('Execute callback function!');
    testFunction();
  }, []);
  const memoFunction = useMemo(() => {
    console.log('Execute memo function!');
    return testFunction();
  }, []);

  console.log('useCallback: ', callbackFunction);
  console.log('useMemo: ', memoFunction);

  return (
    <div>
      <p>Please take a look at the console for the example.</p>
      <p>{ count }</p>
      <button onClick={ () => setCount(count + 1) }>Increase</button>
      <button onClick={ callbackFunction }>Callback</button>
    </div>
  );
}

export default MemoContainer;
```

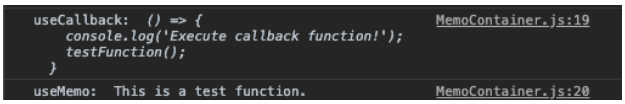
Codefragment 8: Voorbeeld verschil useMemo en useCallback

Allereerst is een teller aangemaakt om het programma verschillende keren opnieuw te laten renderen. Vervolgens is te zien dat de Callback Hook op dezelfde manier is opgebouwd dan de Memo Hook. Wanneer het programma opgestart wordt, is volgende output in de console te herkennen (Figuur 6):



Figuur 6: Output opstart

In eerste instantie is het mogelijk te besluiten dat de Callback Hook de volledige functie teruggeeft en de Memo Hook de waarde uit de functie. Ook is te zien dat de Memo Hook is uitgevoerd, terwijl dit bij de Callback niet het geval is.
Wanneer vervolgens op de knop “Increase” gedrukt wordt, verschijnt volgende output (Figuur 7):



Figuur 7: Output knop "Increase"

In deze output is te zien dat de waardes van de constanten “callbackFunction” en “memoFunction” niet zijn gewijzigd en bijgevolg ook niet uitgevoerd. Hier wordt een gecachte weergave voorgesteld.
Tot slot verschijnt volgende output bij de klik op de knop “Callback” (Figuur 8):

Figuur 8: Output knop "Callback"

Bij de laatste output is te zien dat de functie “callbackFunction” wordt aangeroepen. Deze wordt uitgevoerd, maar vervolgens gebeurt er niets. Dit biedt als extra voordeel dat geen nieuwe rendering wordt uitgevoerd. [23]

5.2.8. useRef

Deze hook wordt gebruikt om een waarde bij te houden. Deze heeft dezelfde mogelijkheid als de State Hook, namelijk een waarde bewaren gedurende de volledige levenscyclus van het component.
Toch is er een verschil tussen de State Hook en Ref Hook. Wanneer de waarde verandert bij een State Hook dan gebeurt een nieuwe rendering. Dit is niet het geval bij de Ref Hook. Ook de syntax is verschillend van de State Hook. Zo wordt de waarde bewaard in het veld “current” van het object.
Onderstaand voorbeeld geeft het gebruik van de Ref Hook aan. (Codefragment 9)

```jsx
import React, { useRef } from 'react';

function RefContainer() {
  const fruit = useRef('Apple');

  return (
    <div>
      <p>{ fruit.current }</p>
      <button onClick={ () => fruit.current = 'Orange' }>Change fruit</button>
    </div>
  );
}

export default RefContainer;
```

Codefragment 9: Voorbeeld useRef

In bovenstaande voorbeeld wordt de waarde aangepast in het “current” veld, maar aangezien deze geen rendering veroorzaakt wordt de waarde in de browser niet aangepast. [24]

5.2.9. useImperativeHandle

Dit systeem wordt zelden rechtstreeks gebruikt, maar wordt meestal achterliggend toegepast in de bibliotheken die zijn geïmporteerd. Omwille van het feit dat React klasse componenten wil afschrijven in de toekomst, moest ook voor de ref-optie een alternatief gezocht worden.

Het onderstaande voorbeeld is een kopie van het voorbeeld van de website van React. Dit omdat het niet mogelijk is een alternatief te bedenken om de werking ervan te bewijzen. Ook is het mogelijk onderstaand voorbeeld te implementeren zonder gebruik te maken van de hook. Verder is het niet mogelijk extra informatie te vergaren in verband met deze functionaliteit. (Codefragment 10 en Codefragment 11) [25] [26]

```jsx
import React from 'react';

// Components
import ImperativeInput from '../components/ImperativeInput';

function ImperativeHandleContainer() {
  const imperativeInputRef = React.createRef();

  return (
    <ImperativeInput ref={ imperativeInputRef }>Click me!</ImperativeInput>
  );
}

export default ImperativeHandleContainer;
```

Codefragment 10: Voorbeeld useImperativeHandle parent component

```jsx
import React, { useRef, useImperativeHandle } from 'react';

function ImperativeInput(props, ref) {
  const inputRef = useRef();

  useImperativeHandle(ref, () => ({
    focus: () => {
      inputRef.current.focus();
    }
  }));

  return <input ref={ inputRef } placeholder={ props.children } />
}

export default React.forwardRef(ImperativeInput);
```

Codefragment 11: Voorbeeld useImperativeHandle

5.2.10. useLayoutEffect

Allereerst is de LayoutEffect Hook identiek aan de Effect Hook. De syntax is bijgevolg volledig hetzelfde, maar het enige verschil tussen beide is de timing. De LayoutEffect Hook wordt synchroon uitgevoerd na elke verandering. Op die manier is er enigszins controle over de timing, wat anderzijds kan resulteren in vertragingen aangezien alles achtereenvolgens uitgevoerd wordt. (Codefragment 2)

Het gebruik van deze hook is uiterst zeldzaam aangezien de voorkeur gaat naar het gebruik van de Effect Hook. Het is dan ook moeilijk enige verschillen in de werking te zien tussen beiden. Wanneer de Effect Hook resulteert in flikkeringen, dan is het aangeraden de LayoutEffect Hook te proberen om dit probleem te verhelpen. [25] [27]

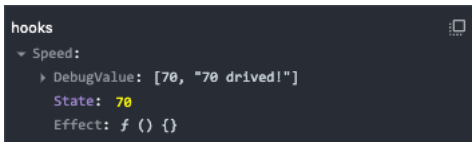
5.2.11. useDebugValue

Een alternatief voor console.log is de DebugValue Hook. Deze wordt gebruikt om een label aan te maken om duidelijkheid te krijgen in de React developer tools. Deze wordt bijgevolg ook enkel gebruikt ter debugging.

Aan deze hook is het mogelijk om twee parameters toe te kennen. Als eerste parameter de waarde, en als tweede een functie ter formatering. Wanneer enkel de eerste wordt meegegeven, dan wordt deze waarde gewoon uitgeprint. Wanneer een tweede parameter wordt meegeven, dan wordt eerst de formatering uitgevoerd en nadien uitgeprint. De tweede parameter wordt gebruikt wanneer het veel resources vraagt om de bewerking uit te voeren. (Codefragment 12 en Figuur 9)
Inmiddels gaat de voorkeur uit naar het gebruik van console.log omwille van de gebruiksvriendelijkheid. [28] [25]

```
useDebugValue(miles);
useDebugValue(miles, mile => mile + ' drived!');
```

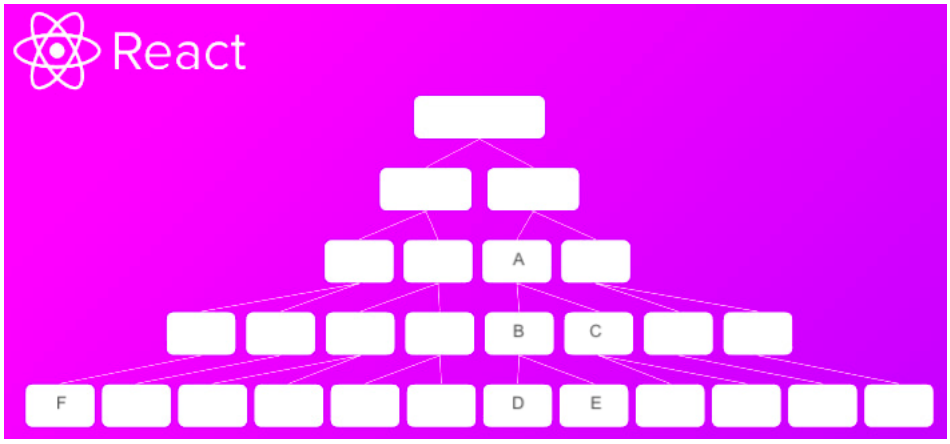
Codefragment 12: Voorbeeld useDebugValue



Figuur 9: Output useDebugValue

5.3. React Context

Context wordt gebruikt om variabelen doorheen de componenten te verspreiden, zonder dat de noodzaak er is om manueel de variabelen door te spelen naar de onderliggende componenten.
Oorspronkelijk wordt data van het bovenliggende component naar het onderliggende component doorgegeven aan de hand van props.



Figuur 10: Structuur componenten in applicatie [29]

In Figuur 10 is een schema te zien hoe de structuur van een applicatie opgebouwd kan zijn. Wanneer variabelen in D en E gebruikt moeten worden, is het de bedoeling de variabelen eerst naar het bovenliggende component te sturen (B) om vervolgens vanuit dit component naar D en E te zenden.
Dit lukt, maar wanneer deze componenten ook in C gebruikt moeten worden is het de bedoeling opnieuw een niveau naar boven te gaan. Hierdoor starten de variabelen vanuit A en worden op deze manier naar de onderliggende componenten genavigeerd. Dit principe is ook nog steeds mogelijk. Maar wat wanneer F ook toegang wenst te hebben tot de variabelen die zich op dat moment in A bevinden? Om dit te bekomen moeten de variabelen van A verplaatst worden naar het root component. Dit wordt na toevoeging van meerdere variabelen niet onderhoudbaar. Voor dit probleem op te lossen bestaat de React Context API. Op die manier moeten de variabelen niet elk niveau worden doorgegeven, maar is het nu mogelijk in willekeurige componenten toegang te krijgen tot de variabelen.

Dit principe kan gebruikt worden om variabelen op te slaan die globaal gebruikt worden. Bijvoorbeeld taal, gebruikersinformatie, … Dit enkel op voorwaarde dat deze variabelen gebruikt moeten worden in verschillende componenten. In het ander geval is het aangeraden gebruik te maken van Component composition. (5.3.1)

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

export const MyContext = React.createContext();

export function MyProvider(props) {
  const [value, setValue] = useState('Day');

  return(
    <MyContext.Provider value={{
      state: value,
      changePeriod:() => setValue(value === 'Day' ? 'Night' : 'Day')
    }}>
      { props.children }
    </MyContext.Provider>
  );
}
```

Codefragment 13: Voorbeeld initialisatie Context

Codefragment 13 geeft aan hoe de initialisatie van de Context API gebeurd. Deze wordt voorzien als wrapper aangezien deze in de root wordt geplaatst omheen de content. Met behulp van props.children is het mogelijk de content te renderen. In deze functie is ook te zien hoe de variabelen doorgestuurd worden en gemanipuleerd.

```
// Import Contexts
import { MyProvider } from './contexts/MyProvider';

ReactDOM.render(
  <MyProvider>
    <Containers />
  </MyProvider>, document.getElementById('root'));
```

Codefragment 14: Voorbeeld wrapper context

Codefragment 14 geeft aan hoe de wrapper wordt geïmplementeerd in de root met behulp van de Provider. Op die manier is het mogelijk vanuit elke component toegang te hebben tot de Context API.
Tot slot wordt aangetoond hoe het mogelijk is de Context API aan te spreken met behulp van de Consumer. In de consumer is het mogelijk een functie te implementeren om de context aan te spreken, zowel de variabelen als de functie. (Codefragment 15) [30]

```
import React from 'react';

// Import Contexts
import { MyContext } from '../contexts/MyProvider';

// assets/css
import '../assets/css/Landscape.scss';

function Landscape() {
  return(
    <MyContext.Consumer>
      {
        context => (
          <div onClick={context.changePeriod} className={ "Landscape__Container--" + context.state }>
            <div className="Landscape__Container__Circle"></div>
          </div>
        )
      }
    </MyContext.Consumer>
  );
}

export default Landscape;
```

Codefragment 15: Voorbeeld aanspreken context

5.3.1. Component composition

In sommige gevallen is het niet aangeraden context te gebruiken omwille van complexiteit die het met zich meebrengt. Wanneer te complex is het aangeraden “component composition” toe te passen.
Een ander alternatief dat toegepast kan worden is overerving, maar wordt zelden tot nooit gebruikt. De voorkeur gaat eerder naar het gebruik van component composition, waarbij het gebruik hieronder geraadpleegd kan worden.

```jsx
import React from 'react';

// Import components
import Card from '../components/Card';

// assets/img
import DummyUser from '../assets/img/DummyUser.png';

function CompositionContainer() {
  return (
    <Card role='Gebruiker'>
      <img width="98" src={ DummyUser } alt="Foto gebruiker" />
      <p>Harold</p>
    </Card>
  );
}

export default CompositionContainer;
```

Codefragment 16: Voorbeeld parent composition component

In Codefragment 16 is de parent component te zien die het Card-component oproept. Hierin is te zien dat de rol wordt meegegeven als variabele aan het component, maar in de parent wordt ook de invulling voor het component beschreven. Anders was er de noodzaak ook de naam mee te sturen zoals de rol, wat na verloop van tijd moeilijk onderhoudbaar is wanneer de diepte van de componenten toeneemt. Wanneer later meerdere variabelen van de parent naar de child moeten toegevoegd worden, zoals bijvoorbeeld de leeftijd en de woonplaats van de gebruiker naar de Card-component, dan is het met component composition eenvoudiger dit meteen in de parent toe te voegen. Wanneer deze variabelen zoals rol moeten meegestuurd worden is dit na verloop van tijd niet overzichtelijk. Dit is zeker het geval wanneer er verschillende levels tussen de parent en de child zitten. [31] [32] [33]
De complexiteit van het component kan op deze manier beschreven worden in de parent. Nadien is het mogelijk de informatie aan te spreken in de child aan de hand van het props-object. (Codefragment 17)

```jsx
import React from 'react';

// assets/css
import '../assets/css/Card.scss';

function Card(props) {
  return (
    <div className="Card__Container">
      <p>{ props.role }</p>
      { props.children }
    </div>
  );
}

export default Card;
```

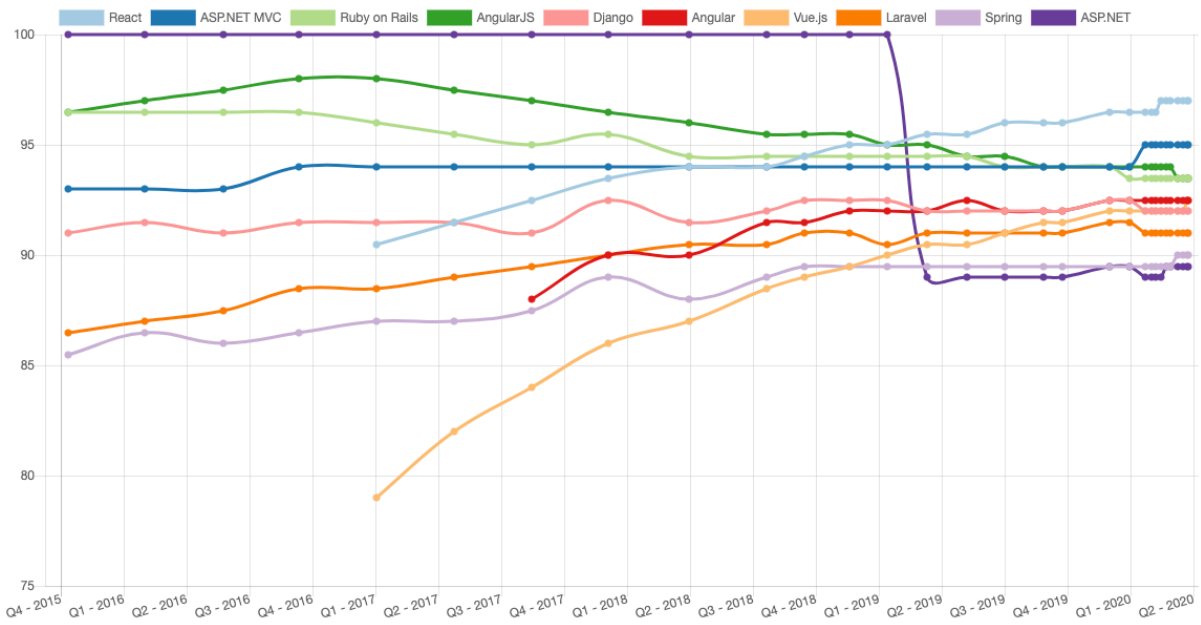
Codefragment 17: Voorbeeld child composition component

Het nadeel van dit principe is dat de logica verzameld wordt op het hoogste niveau. Wat betekent dat hierdoor de complexiteit van dit component stijgt.

5.4. Verschillende frameworks

Binnen deze studie is het onderzoek van React Hooks de hoofdzaak. Zo wordt enerzijds gekeken naar de interne verbetering bij React bij de ontwikkeling van React Hooks, maar anderzijds is het aangeraden de aanpak van andere frameworks eens te bekijken. Specifiek wordt gekeken naar de state management en lifecycle methoden bij andere frameworks.

Hiernaast wordt ook gekeken of de andere hooks voorzien zijn van alternatieven binnen de andere frameworks.
Uiteindelijk is het mogelijk om deze toepassingen naast elkaar te leggen en te vergelijken. Hieruit kan een gepaste conclusie voortvloeien.
Voor de keuze van de gebruikte frameworks wordt onderstaande grafiek in rekening gebracht. Deze beschrijft de populariteit van de verschillende frameworks. (Figuur 11)



Figuur 11: Statistieken populariteit frameworks [34]

Hieruit wordt besloten dat volgende frameworks aan de test worden onderworpen:

- React
    - Functionele componenten
    - Klasse componenten
- AngularJS
- Vue.js

De reden van deze keuze is te danken aan verschillende aspecten. React aangezien dit de bron is van deze studie. AngularJS omdat deze jarenlang een belangrijke functie heeft vervuld op het web, maar de laatste jaren in populariteit is gezakt. De keuze voor Vue.js is te danken aan het feit dat deze ook in populariteit toeneemt, en toch niet behoort tot een groot commercieel bedrijf zoals Facebook en Google, zoals respectievelijk React en AngularJS.
Voor React wordt gebruik gemaakt van functionele componenten, aangezien hierin de hooks worden toegepast, en de klasse componenten. Het is noodzakelijk een vergelijking met klasse componenten te maken aangezien dit de oorspronkelijke manier was voor de ontwikkeling van een component in React.
Om een goed besluit te vormen is een applicatie voorzien voor de COVID-19 pandemie. Op die manier is het mogelijk proefondervindelijk te concluderen en niet enkel te baseren op de theoretische achtergrond. [35]

6. Praktische uitwerking

De bedoeling van dit hoofdstuk is om de voornaamste hooks te bekijken en te vergelijken met de implementatie van de functionaliteiten binnen andere frameworks. Het is namelijk niet zo dat andere frameworks ook voorzien zijn van hooks, maar deze zijn genoodzaakt deze technieken op een andere manier uit te werken.

Als vergelijkingsapplicatie is een actueel onderwerp gebruikt, namelijk de COVID-19 pandemie. Deze applicatie is voorzien van drie pagina's namelijk bevestigde, genezen en sterfgevallen. Elke pagina is voorzien van een dropdown die de verschillende landen aangeeft. Bij het selecteren van een land wordt bijgevolg een GET-request naar de server gestuurd om de data voor dit specifiek land op te halen. Wanneer de data is ontvangen, wordt de grafiek opgevuld met de gegevens die behoren tot één van de drie statussen. Onder de grafiek wordt nog een indicatie gegeven van het aantal inwoners per land, met daaronder nog een indicatie van het aantal opzoekingen. Als laatste staat een knop om het aantal opzoekingen te manipuleren en de teller naar beneden te halen.
Op die manier zijn de belangrijkste hooks gebruikt in één applicatie. Hierbij gaat het over de volgende hooks:

- useState
- useEffect
- useReducer
- useCallback
- Custom Hook
- useContext

De overige hooks zijn buiten beschouwing gelaten omwille van het beperkte gebruik of omdat ze framework afhankelijk zijn.
De state management wordt bekeken aan de hand van useState, die de lokale variabelen binnen een component beheert. Voor de globale variabelen wordt useContext gebruikt.

Het is de bedoeling in onderstaande vergelijkingen om de React functionele componenten te gebruiken als maatstaf naar de andere frameworks om na te gaan welke methode voordeliger is.

6.1. React Hook: useState

Eén van de meest gebruikte hooks is useState en neemt bij ontwikkeling van een applicatie een belangrijke rol in. Aan de hand van deze React Hook is het mogelijk om variabelen lokaal bij te houden.
Deze React Hook is toegepast om onder andere de code van het land bij te houden. Dit omwille van het feit dat deze in verschillende functies gebruikt wordt. Naast de code van het land wordt de data voor de grafiek ook tijdelijk opgeslagen.

6.1.1. React – functionele componenten

In de werking van de hook wordt verder ingegaan in 5.2.1 useState. Om deze hook te gebruiken is het noodzakelijk deze te importeren, bijgevolg te declareren en initialiseren om tot slot hem te kunnen oproepen en een waarde te kunnen toekennen. (Codefragment 1)

6.1.2. React – klasse componenten

Aangezien de klasse overerft van React.Component, is het mogelijk om gebruik te maken van het “this”-object en is intern voorzien van een veld state die een object bijhoudt met verschillende variabelen.

```
export default class ConfirmedGraphContainer extends Component {
  static contextType = CovidContext;

  constructor(props) {
    super(props);
    this.state = {
      countryCode: null,
      prevCountryCode: null,
      chartData: null,
      showChart: false,
      data: [],
      axes: [
        { primary: true, type: 'utc', position: 'bottom' },
        { type: 'linear', position: 'left' }
      ],
      population: 0,
      count: 0
    };

    ...
```

Codefragment 18: State bij React klasse component

Bovenstaande afbeelding geeft aan hoe het mogelijk is in de constructor een standaard waarde toe te kennen. Nadien is het ook mogelijk om deze waarde te manipuleren aan de hand van de functie “setState”. Aan deze functie wordt als parameter een object toegediend. (Codefragment 18)
De waarde uiteindelijk aanspreken is mogelijk aan de hand van bijvoorbeeld: this.state.countryCode.

6.1.3. Vue

Een Vue applicatie wordt opgebouwd aan de hand van “new Vue({ … })”, waarbij opties worden meegegeven. De filosofie dat Vue hanteert is SPA, of beter bekend als Single Page Application, wat aangeeft dat het volledige project opgebouwd wordt aan de hand van een root Vue instantie.
Via de CLI, of Command Line Interface, is het mogelijk een nieuwe applicatie aan te maken. Hierbij wordt een startversie aangemaakt. Tussen de bestanden die aangemaakt zijn, is main.js terug te vinden. In dit bestand is deze root Vue instantie terug te vinden en deze refereert vervolgens naar de router folder waarin een index.js bestand staat. Hierin worden de views toegekend aan URL’s.
Uit deze views/componenten worden objecten teruggestuurd die worden geïmporteerd in de Vue instantie in main.js. Op die manier kan een applicatie in Vue aangemaakt worden met gebruik van slechts één Vue instantie.

Eén van deze opties in het object dat teruggestuurd wordt uit een view/component is data. Deze optie bevat een functie dat een object exporteert met hierin de verschillende variabelen die gebruikt worden in deze view.

```
export default {
  name: "app",
  components: {
    Header,
    MainContainer
  },
  data() {
    return {
      isCountriesSet: false
    };
  },

  ...
```

Codefragment 19: State bij Vue

Bovenstaande afbeelding toont een deel van het object dat geëxporteerd wordt. Hierin is de syntax terug te vinden voor het bijhouden van lokale variabelen in een component. (Codefragment 19)
Om vervolgens deze data te manipuleren uit een functie, kan dit aan de hand van het “this”-object. In het bovenstaande voorbeeld zou dit op volgende manier moeten: this.isCountriesSet = true.
Vue gebruikt een HTML-gebaseerde template syntax om de onderliggende data voor te stellen in de DOM. Dit is mogelijk aan de hand van de “Mustache” syntax, wat het volgende voorstelt wanneer toegepast op bovenstaande voorbeeld: {{ isCountriesSet }}. Hier is het niet nodig om het “this”-object voor te plaatsen. [36]

6.1.4. Angular

De opbouw van een component in Angular is niet functioneel, maar hier wordt gebruik gemaakt van klasses. Hierbij wordt bovenaan de klasse de variabelen gedeclareerd en wanneer nodig ook geïnitialiseerd. Aangezien Angular gebruik maakt van Typescript dient het type van de variabele te worden aangegeven. Als extra mogelijkheid is het mogelijk om de variabelen publiekelijk of privaat te maken in de klasse. Hierdoor kan de toegankelijkheid van deze variabele in een andere klasse vermeden worden. Het is wel mogelijk private velden te gebruiken in de view van deze klasse.

```
export class ConfirmedGraphContainer implements OnInit {
  public countries: Array<Object>;
  public showChart: Boolean = false;
  private countryCode: string = 'BE';
  public chart: any = [];
  private cases: Array<Number> = [];
  private dates: Array<any> = [];
  public population: number;
  public count: number = 0;
  private countInstance: Count;

  ...
```

Codefragment 20: State bij Angular

In bovenstaande afbeelding is een voorbeeld terug te vinden hoe de declaratie en initialisatie gebeurd binnen een klasse. (Codefragment 20)
Om uiteindelijk de waarde aan te spreken wordt de “Mustache” syntax gebruikt op identieke manier als bij Vue.

6.1.5. Conclusie

Bovenstaande voorbeelden geven nu de verschillende manieren weer voor het gebruik van de lokale variabelen binnen een component.
Uit deze vergelijking is te besluiten dat bij React functionele componenten een omslachtige manier wordt gebruikt voor declaratie/initialisatie. Dit is een aparte syntax dat bij de anderen niet gekend moet zijn.
Bij React klasse componenten moet een aparte functie gebruikt worden voor het manipuleren van een variabele dat overbodig is ten opzichte van Angular en Vue.
Vue biedt een eenvoudige manier voor het aanspreken en manipuleren van deze variabelen.
Tot slot heeft Angular een overzichtelijke manier voor het gebruik van deze variabelen net als Vue, maar brengt als nadeel met zich mee dat de bestanden voor de view en controller van elkaar gescheiden zijn waardoor het omslachtig wordt wanneer aan verschillende componenten gelijktijdig gewerkt wordt.

6.2. React Hook: useEffect

Naast de useState Hook is de useEffect Hook niet weg te denken. Op deze manier is het mogelijk om lifecycle methoden toe te voegen aan een component. Zo is het onder andere mogelijk data van een API op te halen alvorens de inhoud van de pagina op het scherm wordt afgebeeld, of anders gezegd gemount is.
Deze methode wordt gebruikt om de landen op te halen die vervolgens in de dropdown worden geplaatst. Het is noodzakelijk hiervoor de useEffect Hook te gebruiken aangezien het vereist is de dropdown meteen opgevuld te hebben wanneer de inhoud van de pagina geladen is.

6.2.1. React – functionele componenten

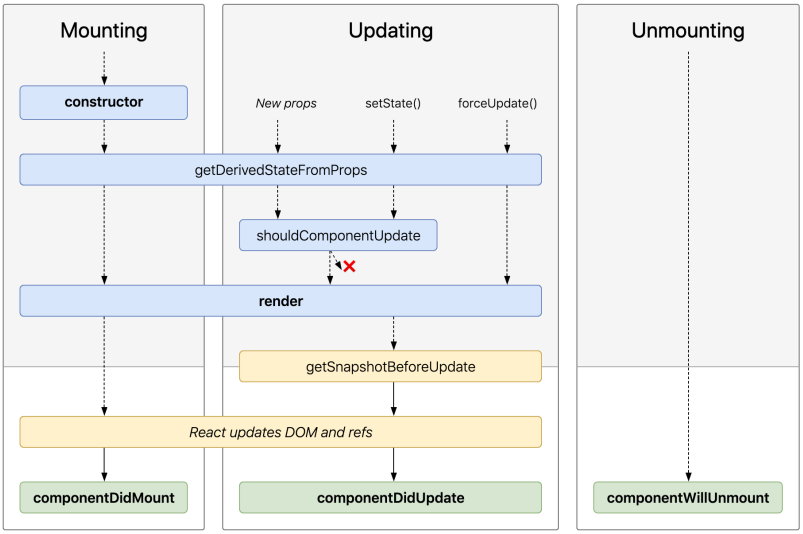
De syntax voor deze hook is reeds vermeld in hoofdstuk 5.2.2 useEffect. Hiervoor dient useEffect geïmporteerd te worden, en nadien de useEffect op te roepen met als parameter de functie met de code die uitgevoerd dient te worden bij re-render. Ook is de mogelijkheid aanwezig tot het toevoegen van een tweede parameter, dat een array voorstelt van waardes. De hook wordt dan enkel uitgevoerd wanneer één van deze waarden wijzigt. Zo worden oneindige loops vermeden. (Codefragment 2: Voorbeeld useEffect)
In onderstaande afbeelding is de toepassing van de useEffect hook te zien gebruikt binnen de vergelijkingsapplicatie. (Codefragment 21)

```
useEffect(() => {
  if (!covidContext.countries) {
    async function getCountriesAsync() {
      let countries = await getCountries();
      covidContext.setCountries(countries);
    }
    getCountriesAsync();
  }
}, [covidContext]);
```

Codefragment 21: Lifecycle methode bij React functioneel component

6.2.2. React – klasse componenten

Bij klasse componenten wordt ook gebruik gemaakt van lifecycle methoden, enkel bestaan hier verschillende methoden voor. Elke methode wordt op een ander moment in de levenscyclus van het component uitgevoerd.



Figuur 12: Schema lifecycle methodes React klasse component [37]

In bovenstaande afbeelding is een overzicht te vinden van de verschillende lifecycle methoden en wanneer deze exact worden uitgevoerd bij mounting, update en unmount. (Figuur 12: Schema lifecycle methodes React klasse component)

In onderstaande afbeelding is een voorbeeld van een lifecycle methode terug te vinden die gebruikt wordt ter vervanging van de useEffect Hook. Deze moeten niet expliciet geïmporteerd worden aangezien deze worden aangeleverd uit React.Component, waarvan de klasse overerft. (Codefragment 22)

```
async componentDidMount() {
  if (!this.context.countries) {
    this.context.setCountries(await getCountries());
  }
}
```

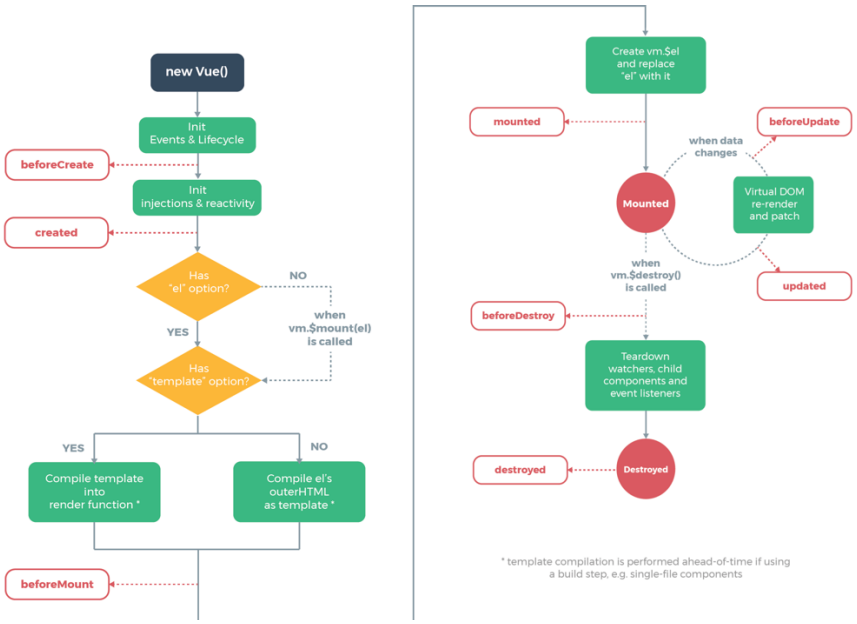
Codefragment 22: Lifecycle methode bij React klasse component

Ten opzichte van de useEffect hook wordt hetzelfde bereikt met vier lijnen minder code. Bij React functioneel component moeten eerst nog een aantal checks uitgevoerd worden en een omweg voorzien worden om asynchrone code synchroon te laten uitvoeren. Deze

functionaliteiten worden achterliggend bij React klasse componenten uitgevoerd, waardoor de ontwikkelaar hier weinig obstakels door ondervindt.

6.2.3. Vue

Ook bij Vue is het gebruik van lifecycle methodes niet weg te denken. Hier wordt ook gebruik gemaakt van verschillende functies die elk op een ander moment uitgevoerd worden. Dit om acties op de juiste plaats uit te voeren.



Figuur 13: Schema lifecycle methodes Vue [38]

In bovenstaande afbeelding is een schema dat de levenscyclus van een component voorstelt. Aan de hand van dit schema is het mogelijk een gepaste functie te kiezen die het best aanleunt tegen jouw doeleinden. (Figuur 13)
Voor de vergelijkingsapplicatie volstond de mounted functie voor het ophalen van de landen. De volgende afbeelding geeft dezelfde functionaliteit weer die bereikt wordt aan de hand van de useEffect Hook.

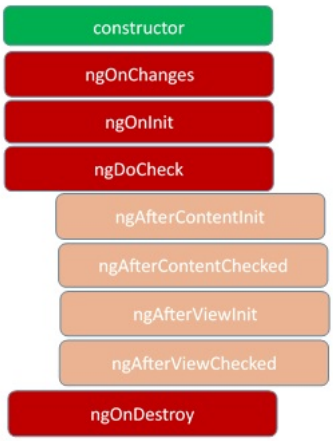
```
mounted: async function() {
  Vue.prototype.$countries = await getCountries();
  this.isCountriesSet = true;
}
```

Codefragment 23: Lifecycle methode bij Vue

Het is reeds duidelijk dat deze functionaliteit korter geschreven kon worden bij React klasse component als bij React functioneel component. Maar dit was buiten Vue gerekend, aangezien hierbij nog een lijn gespaard blijft buiten de React klasse component.

6.2.4. Angular

Tot slot maakt Angular ook gebruik van lifecycle methoden. In onderstaande afbeelding kan het schema geraadpleegd worden welke lifecycle methodes bestaan, afgebeeld in de volgorde waarin deze worden uitgevoerd. (Figuur 14)



Figuur 14: Schema lifecycle methodes Angular [39]

Voor de applicatie wordt enkel gebruik gemaakt van de ngOnInit()-functie. Dit volstaat voor het ophalen van de verschillende landen bij opstart van de applicatie.

```
async ngOnInit() {
  await (await this.covidService.getCountries()).subscribe(res =>
    this.globalsService.setCountries(res)
  );
}
```

Codefragment 24: Lifecycle methode bij Angular

Omwille van het feit dat bij Angular gebruik gemaakt wordt van observables, wordt hierdoor de complexiteit van de code ten opzichte van React en Vue groter. In React en Vue kan de informatie behaald worden door het oproepen van de functie zonder hier een subscribe op uit te voeren. (Codefragment 24)

6.2.5. Conclusie

Wanneer het gebruik van de lifecycle methoden met elkaar wordt vergeleken, is te zien dat bij React Hooks een omweg dient gemaakt te worden om op die manier asynchrone code synchroon uit te voeren. Hierdoor vergroot de code aanzienlijk.
De syntax van de hook ten opzichte van de andere manieren is complexer. De andere frameworks maken gebruik van een functionele syntax, terwijl de hook gebruik maakt van een callback functie.
Aangezien de functies een gelijke actie vervullen, is te besluiten dat de observable van Angular de code complexer maakt dan bij React klassen en Vue.

6.3. React Hook: useReducer

Een andere hook waarbij de functionaliteit vergeleken wordt, is de useReducer hook. Deze hook zorgt ervoor dat een variabele aangepast wordt naargelang de actie die wordt aangegeven.

Om de functionaliteit van deze hook te testen, wordt een telling in de applicatie toegevoegd. Hierbij wordt per opzoeking de teller met één vermeerderd, en wanneer op de voorziene knop wordt gedrukt is er de mogelijkheid de teller met één te verminderen.

6.3.1. React – functionele componenten

Het gebruik van deze hook wordt in detail uitgelegd in 5.2.5 useReducer. De logica van de functie is verplaatst naar een extern bestand, waardoor het mogelijk is deze te hergebruiken. (Codefragment 25)

```
export function countReducer(state, action) {
  if (action.type === 'increment') {
    return { count: state.count + 1 };
  }
  else if (action.type === 'decrement' && state.count !== 0) {
    return { count: state.count - 1 };
  }
  return state;
}
```

Codefragment 25: Reducer functie bij React functioneel component

Vervolgens wordt de useReducer hook geïmporteerd van de React library, en ook de countReducer()-functie wordt geïmporteerd vanuit het extern bestand.
Nadien gebeurt de declaratie en initialisatie, waarbij de code te raadplegen is in onderstaande codefragment. (Codefragment 26)

```
const [state, dispatch] = useReducer(countReducer, { count: 0 });
```

Codefragment 26: Reducer declaratie bij React functioneel component

Tot slot kan de uitkomst van de functie geraadpleegd worden aan de hand van de state-variabele. Om vervolgens een actie uit te voeren dient de dispatch()-functie opgeroepen te worden. Hierbij is het noodzakelijk als parameter een object met de actie mee te geven.

6.3.2. React – klasse componenten

Het principe van de useReducer is bij klasse componenten niet gekend. In dit geval is het de bedoeling zelf een alternatief/omweg te voorzien om dergelijke functionaliteit te bekomen. In het bestand Helper.js is een functie voorzien die deze functionaliteit bevat.
(Codefragment 27)

```
export function countViews(count, action) {
  if (action === 'increment') {
    count++;
  }
  else if (action === 'decrement' && count !== 0) {
    count--;
  }
  return count;
}
```

Codefragment 27: Reducer functie bij React klasse componenten

Nadien is het de bedoeling in de container, het bestand waarin de view beschreven staat, deze functie te importeren vanuit Helper.js. Nadien wordt deze functie op gepaste momenten opgeroepen aan de hand van volgende syntax:
bv. “countViews(this.state.count, ‘increment’)”

Aan de hand van de setState()-functie is het nadien mogelijk de uitkomst van deze functie op te slaan in de count-variabele.

6.3.3. Vue

Bij Vue is het principe van de useReducer hook niet gekend. Ook hier was het noodzakelijk een alternatief te vinden. Uiteindelijk bleek het concept van mixins een goede vervanging te zijn.
Een mixin is een flexibele manier om code te hergebruiken. In dit object wordt ook gebruik gemaakt van component opties zoals data, computed, methods, components, … Zoals het woord zegt worden de component opties van de mixin gemixt met de component opties van de view. [40]
Ter vervanging van de useReducer, wordt hierbij gebruik gemaakt van de computed optie van Vue.

```
var CountMixin = {
  data() {
    return {
      count: 0
    };
  },
  computed: {
    increaseCount: {
      set: function(value) {
        this.count = this.count + value;
      },
      get: function() {
        return this.count;
      }
    },
    decreaseCount: {
      set: function(value) {
        if (this.count !== 0) {
          this.count = this.count - value;
        }
      },
      get: function() {
        return this.count;
      }
    }
  }
};
```

Codefragment 28: Reducer functie bij Vue

In deze computed optie zijn twee variabelen te herkennen, namelijk increase- en decreaseCount met elk voorzien van een getter- en setter-functionaliteit. In deze getters en setters wordt de count-variabele gemanipuleerd. (Codefragment 28)
Uiteindelijk in de view wordt de mixin geïmporteerd en is bijgevolg aanspreekbaar aan de hand van bv. “this.increaseCount = 1”. Aan de hand van deze manier is het mogelijk om de teller in plaats van telkens met één stap, met meerdere stappen te verhogen. Wanneer bijgevolg telkens verhoogd wordt met de waarde 2, dan kan dit aan de hand van “this.increasCount = 2”. Dit brengt als voordeel mee dat hiervoor geen extra code moet geschreven worden. Hierdoor is deze manier multifunctioneel. De waarde kan uiteindelijk bekeken worden aan de hand van de count-variabele. (Codefragment 29)

```
export default {
  name: "Confirmed",
  mixins: [PopulationMixin, CountMixin],
  data() {
    return {
      countryCode: "BE",
      chartData: null,
      showChart: false,
      population: 0,
      count: 0
    };
  },

  ...
```

Codefragment 29: Importeren Mixin in Vue

6.3.4. Angular

Om deze functionaliteit te integreren bij Angular dient een gelijkaardige manier gehandhaafd te worden als bij React klasse componenten. Hierbij dient een klasse Count aangemaakt te worden, waarin een constructor is voorzien samen met twee functies voor het op- en aftellen. (Codefragment 30)

```
export class Count {
  private count: number;

  constructor() {
    this.count = 0;
  }

  increaseCount() {
    this.count = this.count + 1;
    return this.count;
  }

  decreaseCount() {
    if (this.count > 0) {
      this.count = this.count - 1;
    }
    return this.count;
  }
}
```

Codefragment 30: Reducer klasse bij Angular

Nadien is het noodzakelijk de klasse te integreren in de klasse van de view. Dit is mogelijk door een instantie van de Count klasse aan te maken in de constructor van de klasse van de view. Dit kan aan de hand van “this.countInstance = new Count()”, waarbij countInstance een private variabele binnen de klasse van de view is. Via deze instantie is het mogelijk om toegang te krijgen tot de functies binnen de Count klasse. Om deze functie aan te roepen wordt volgende syntax gebruikt: “this.count = this.countInstance.increaseCount()”. Om nu de variabele weer te geven in de view is het mogelijk gebruik te maken van de publieke count variabele die aanspreekbaar is aan de hand van de “Mustache” syntax: {{ count }}.

6.3.5. Conclusie

Na het bekijken van deze voorbeelden is te besluiten dat de functionaliteit van deze hook overbodig is. Het is mogelijk aan de hand van een andere manier, al dan niet eenvoudiger, hetzelfde doel te bereiken.

De React functionele component, waarbij de useReducer hook gebruikt wordt, bevat een syntax die een omslachtige structuur vertoond. Hierdoor wordt de hook ingewikkeld.
Vervolgens bij React klasse componenten is een gelijkaardige manier nagemaakt, waarbij de manier van aanroepen van de functie eenvoudiger en vanzelfsprekender is dan bij React functionele componenten.
De methode van Vue is eenvoudiger eens het concept van de mixin duidelijk is. Nadien is het mogelijk eenvoudig herbruikbare code te schrijven.
Tot slot is de methode van Angular het meest overzichtelijk wanneer reeds ervaring aanwezig is binnen objectgeoriënteerde programmeertalen.
De code voor het bekomen van deze functionaliteit is het meest beperkt bij React klasse componenten. Hiernaast is het concept van de mixins interessant aangezien deze veel mogelijkheden biedt. Eén van deze mogelijkheden wordt in de vergelijking van de custom hook verder duidelijk.

6.4. React Hook: useCallback

Het concept van hooks is om functioneel programmeren mogelijk te maken. Hierdoor is er een obstakel aanwezig aangezien het niet mogelijk is een functie te schrijven in een functie. Om dit probleem te verhelpen is de Callback Hook ontstaan.
Het gebruik van deze hook is terug te vinden wanneer de waarde van de dropdown aangepast wordt.

6.4.1. React – functionele componenten

De exacte werking wordt concreet toegelicht in hoofdstuk 5.2.6 useCallback. De eerste stap om gebruik te kunnen maken van deze hook is door deze te importeren via de React-bibliotheek. In deze bibliotheek staan de verschillende hooks inmiddels allemaal gedefinieerd, waaronder ook de Callback Hook.

```
const changeValueDropdown = useCallback(async country => {
  if (country !== 'default') {
    dispatch({ type: 'increment' });
    let countryValues = country.split(',');
    setCountryCode(countryValues[1]);
    let confirmedStats = await getByCountry(countryValues[0], 'confirmed');
    let tempChartData = [];
    confirmedStats.data.map(confirmedStat => {
      return tempChartData.push({ x: new Date(confirmedStat.Date), y: confirmedStat.Cases });
    });
    setChartData(tempChartData);
    setShowChart(confirmedStats.data.length !== 0);
  }
  else setShowChart(false);
}, []);
```

Codefragment 31: Callback hook bij React functioneel component

In bovenstaand codefragment is het gebruik van deze hook terug te vinden. Deze is voorzien van twee parameters, namelijk de functie die uitgevoerd wordt wanneer de hook wordt opgeroepen en een array van afhankelijkheden. De waarden in de array stellen de variabelen voor die gebruikt worden in de callback.
De variabele “changeValueDropdown” bevat uiteindelijk een gememoriseerde weergave van de callback. Deze wijzigt enkel wanneer een afhankelijkheid in de array, die als tweede parameter is voorzien, wijzigt. [22] (Codefragment 31)

De uiteindelijke invulling van deze functie gaat als volgt te werk en is ook van toepassing op de ander frameworks. Allereerst wordt de teller met één verhoogt die het aantal zoekopdrachten gaat bijhouden per sessie (Reducer Hook). Vervolgens wordt de waarde van de dropdown verkregen en ontleedt, om vervolgens de geselecteerde code binnen het component te bewaren (State Hook). Nadien wordt de data voor de grafiek opgehaald afkomstig van een API, of beter bekend als een Application Programming Interface. Dit stelt een verzameling van definities voor waardoor het mogelijk is een communicatie tussen twee programma's te doen ontstaan. Dit is meestal ook de manier van communiceren tussen de frontend en backend. Uiteindelijk wordt deze data in een correcte vorm gegoten waardoor deze leesbaar wordt voor de grafiek. Deze data wordt ook lokaal binnen het component opgeslagen, samen met een extra veld die aangeeft of de grafiek al dan niet wordt weergegeven (State Hook). [41]

6.4.2. React – klasse componenten

Binnen klasse componenten is de filosofie om een component op te bouwen aan de hand van klassen, waardoor het eenvoudig is een functie toe te kennen aan deze klasse. In onderstaande codefragment is te zien hoe een functie wordt toegekend aan een klasse. Voor deze methode is geen import nodig. (Codefragment 32)

```
export default class ConfirmedGraphContainer extends Component {

  async changeValueDropdown(country) {
    if (country !== 'default') {
      this.setState({ count: countViews(this.state.count, 'increment') });
      let countryValues = country.split(',');
      this.setState({ countryCode: countryValues[1] });
      let confirmedStats = await getByCountry(countryValues[0], 'confirmed');
      let tempChartData = [];
      confirmedStats.data.map(confirmedStat => {
        return tempChartData.push({ x: new Date(confirmedStat.Date), y: confirmedStat.Cases });
      });
      this.setState({
        data: [
          {
            label: 'Confirmed covid-19 patients',
            data: tempChartData
          }
        ],
        showChart: confirmedStats.data.length !== 0
      });
    }
    else this.setState({ showChart: false });
  }

  ...
```

Codefragment 32: Functie bij React klasse component

De opbouw van de functie loopt gelijkaardig als de manier uitgelegd in 6.4.1.

6.4.3. Vue

Zoals eerder vermeld in 6.3.3 wordt Vue opgebouwd aan de hand van een object voorzien van component opties. Eén van deze opties is "methods", dat bijgevolg opnieuw een object is. Dit object bevat variabelen die elk een functie bevatten en hierin worden ook de lifecycle methoden gedefinieerd.

```
methods: {
  changeValueDropdown: async function(country) {
    if (country !== "default") {
      this.increaseCount = 1;
      let countryValues = country.split(",");
      this.countryCode = countryValues[1];
      let stats = await getByCountry(countryValues[0], "confirmed");
      let chartData = {};
      stats.data.map(stat => {
        if (stat.Country !== "") {
          return (chartData[stat.Date.split("T")[0]] = stat.Cases);
        }
      });
      this.chartData = chartData;
      this.showChart = stats.data.length !== 0;
    } else this.showChart = false;
  }
},
```

Codefragment 33: Functie bij Vue

De opbouw van de functie loopt gelijkaardig als de manier uitgelegd in 6.4.1.

6.4.4. Angular

De manier voor het definiëren van een functie is sterk gelijkend aan deze van React klasse componenten. Hierbij wordt ook gebruik gemaakt van een klasse waarbinnen het mogelijk is verschillende functies te definiëren. (Codefragment 34)

```
async changeValueDropdown(country) {
  if (country !== "default") {
    this.count = this.countInstance.increaseCount();
    let countryValues = country.split(",");
    this.countryCode = countryValues[1];
    await (await this.covidService.getByCountry(countryValues[0], "confirmed")).subscribe(res => {
      this.createChart(res);
    });
    let populationInstance = new Population(this.populationService);
    this.population = await populationInstance.getPopulation(this.countryCode);
  } else this.showChart = false;
}
```

Codefragment 34: Functie bij Angular

De opbouw van de functie loopt gelijkaardig als de manier uitgelegd in 6.4.1.

6.4.5. Conclusie

Angular en React klasse componenten hebben een soortgelijke oplossing voor het schrijven van een functie, waarbij deze binnen de klasse beschreven worden. Deze methode zorgt voor een overzichtelijke manier.
Bij React functionele componenten daarentegen wordt gebruik gemaakt van de Callback Hook. Dit is een functie waaraan een functie wordt meegegeven en hierdoor een omslachtig gevoel nalaat.
Tot slot bij Vue is de manier moeilijk vergelijkbaar aangezien hier een andere aanpak aanwezig is. In dit concept worden deze toegekend aan de variabelen van een object.
De functies hebben over de verschillende frameworks allen dezelfde werking, maar wanneer naar de lengte van de functies gekeken wordt kan besloten worden dat Angular erin geslaagd is de functie zo kort mogelijk te houden.

6.5. React Hook: Custom Hook

Het doel van een Custom Hook is om herbruikbare code af te zonderen. Maar wat is bijgevolg het verschil tussen een functie en een Custom Hook?
In een Custom Hook is het mogelijk gebruik te maken van andere hooks voorzien in de React-bibliotheek.
In dit voorbeeld wordt de Custom Hook gebruikt voor het ophalen van het aantal inwoners wanneer een land wordt geselecteerd. Deze waarde wordt vervolgens op het scherm afgebeeld.

6.5.1. React – functionele componenten

Allereerst is het noodzakelijk om de functie voor deze Custom Hook te voorzien. In onderstaande afbeelding is het mogelijk deze hook terug te vinden. Binnen deze Custom Hook wordt gebruik gemaakt van de State- en Effect Hook, waarbij de state wordt gebruikt om de waarde te onthouden en terug te sturen. De Effect Hook daarentegen wordt gebruikt voor het uitvoeren van de code wanneer een update plaatsvindt. Deze code wordt enkel uitgevoerd wanneer de variabele “countryCode” een wijziging ondervindt. (Codefragment 35)

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

// rest
import { getPopulation } from '../../rest/data/PopulationData';

function useGetPopulation(countryCode) {
  const [population, setPopulation] = useState(0);

  useEffect(() => {
    async function fetchPopulation() {
      if (countryCode) {
        let population = await getPopulation(countryCode);
        setPopulation(population);
      }
    }
    fetchPopulation();
  }, [countryCode]);

  return population;
}

export default useGetPopulation;
```

Codefragment 35: Custom Hook bij React functioneel component

Nadien is het noodzakelijk deze hook te importeren in de container, en deze vervolgens binnen het functioneel component bovenaan aan te roepen. Wanneer deze verder in het functioneel component wordt aangeroepen, wordt een foutmelding gegenereerd.
Verdere details in verband met deze hook zijn terug te vinden in 5.2.3 Custom Hooks.

6.5.2. React – klasse componenten

Bij klasse componenten is het niet mogelijk deze hook te gebruiken aangezien deze specifiek toegankelijk is voor React functionele componenten. Omwille van deze reden is het noodzakelijk een omweg te voorzien.

```
async componentDidUpdate(prev, current) {
  if ((current.countryCode !== current.prevCountryCode) && current.countryCode) {
    this.setState({
      population: await getPopulationByCountry(this.state.countryCode),
      prevCountryCode: current.countryCode
    });
  }
}
```

Codefragment 36: Update lifecycle methode bij React klasse component

De beste manier om tot een gelijkaardige oplossing te komen is door gebruik te maken van componentDidUpdate()-functie, wat een lifecycle methode voorstelt. Nadien dient manueel geverifieerd te worden indien de “countryCode” geüpdatet is, om tot slot de nodige code uit te voeren. De code die mogelijk is om hier af te zonderen is het ophalen van het aantal inwoners. Verder code afzonderen is niet mogelijk aangezien de state gebonden is aan de klasse. (Codefragment 36)

6.5.3. Vue

Om bij Vue tot een gelijkaardige oplossing te komen wordt een mixin gebruikt. Het principe van een mixin wordt uitgelegd in 6.3.3 Vue.

```
import { getPopulationByCountry } from "@/helpers/Helper";

var PopulationMixin = {
  data() {
    return {
      population: 0
    };
  },
  watch: {
    chartData: async function() {
      this.population = await getPopulationByCountry(this.countryCode);
    }
  }
};

export default PopulationMixin;
```

Codefragment 37: Mixin watcher bij Vue

Een andere component optie dat bij Vue is ingebouwd is de “watch”. Deze component optie bevat een object met keys die variabelen voorstellen in een view. Wanneer deze variabele een wijziging ondervindt, dan wordt de waarde van deze key uitgevoerd. In Codefragment 33 is de functie in Vue terug te vinden. Wanneer uiteindelijk de waarde van “chartData” wordt aangepast, dan wordt bijgevolg deze watch aangeroepen. Hierdoor wordt de functie uitgevoerd voor het ophalen van het aantal inwoners voor dat specifiek land.
Het bijkomende voordeel van een mixin is de mogelijkheid om gebruik te maken van variabelen in een view. De variabele “population” bij data stelt een standaard waarde voor om “null” of “undefined” waardes te vermijden.

6.5.4. Angular

Omwille dat bij Angular de implementatie niet lukte op de manier zoals in bovenstaande voorbeelden is toegepast, was de noodzaak aanwezig om de logica voor de populatie af te zonderen naar een aparte klasse. Aan de constructor van de klasse wordt de service instantie meegegeven die het mogelijk maakt interactie met de API te voeren. De functie van deze klasse voor het ophalen van het aantal inwoners wordt meteen hierna opgeroepen. (Codefragment 34)

6.5.5 Conclusie

Het idee achter het concept van de Custom Hook is zeer voordelig in een aantal gevallen en bespaart enorm op dubbele code.
Bij React klasse componenten en Angular is het mogelijk een omweg te voorzien, maar maken niet meteen gebruik van de filosofie van de Custom Hook.
Vue daarentegen met het concept van de mixin volgt een stuk van de visie van de Custom Hook om dubbele code te vermijden met de mogelijkheid gebruik te maken van lifecycle methoden en dergelijke. Maar Vue gaat hier nog verder op in om nog meer te besparen op de hoeveelheid code.
Aangezien elk een andere manier gebruikt om de Custom Hook tot stand te brengen is het mogelijk te vergelijken op de lengte, maar dit zou niet in verhouding staan met de eenvoud en herbruikbaarheid.

6.6. React Hook: useContext

De State Hook wordt gebruikt om een waarde bij te houden binnen een component. Wanneer deze waarde verder dient gebruikt te worden in een ander component, dan wordt deze als attribuut aan de HTML-tag meegegeven. Als het component verschillende levels lager zit dan het huidige component, dan moet deze waarde voortdurend doorgegeven worden.

```
function App() {
  return <List theme="dark" />;
}

function List(props) {
  return (
    <div>
      <ListItem theme={ props.theme } />
    </div>
  );
}

function ListItem(props) {
  return <Button theme={ props.theme } />;
}
```

Codefragment 38: Variabele verschillende componenten doorgeven

In bovenstaande afbeelding is te zien hoe de variabele verschillende levels wordt doorgegeven. In dit geval wordt het thema doorgegeven, maar het thema gaat uiteindelijk in bijna alle componenten in de applicatie aanwezig zijn. Wanneer dit het geval is, is het gebruik van de Context API aangeraden. Hierdoor is het mogelijk het thema globaal toegankelijk te maken. (Codefragment 38)
In de applicatie wordt de Context API gebruikt om de landen globaal bij te houden. Nadien is het mogelijk deze te gebruiken in elke container.

6.6.1. React – functionele componenten

De manier waarop de Context API geïntegreerd wordt binnen de applicatie is te raadplegen in hoofdstuk 5.2.4 useContext.

```
const covidContext = useContext(CovidContext);

useEffect(() => {
  if (!covidContext.countries) {
    async function getCountriesAsync() {
      let countries = await getCountries();
      covidContext.setCountries(countries);
    }
    getCountriesAsync();
  }
}, [covidContext]);
```

Codefragment 39: Aanpassen van het Context-object bij React functioneel component

In bovenstaande codefragment is de opstelling van de Context API reeds gebeurd, en is het de bedoeling de Context te declareren en vervolgens te initialiseren. Het initialiseren gebeurd in de Effect Hook door de zelfgemaakte set()-functie te gebruiken. (Codefragment 39)

```
export function CovidProvider(props) {
  const [countries, setCountries] = useState(null);

  return (
    <CovidContext.Provider value={{
      countries,
      setCountries: (value) => setCountries(value)
    }}>
      { props.children }
    </CovidContext.Provider>
  );
}
```

Codefragment 40: Context provider bij React functioneel component

Vervolgens is de opzet van de provider terug te vinden, waarin de landen worden opgeslagen. In deze provider worden deze bijgehouden aan de hand van de State Hook. Als attribuut wordt aan de provider een object meegegeven waarbij de landen kunnen geraadpleegd en toegekend worden. Het toekennen gebeurd met behulp van een functie die vervolgens de waarde opslaat in de State Hook. (Codefragment 40)

6.6.2. React – klasse componenten

Het gebruik van de Context API bij klasse componenten is sterk gelijkend aan deze van de functionele componenten.

```
export default class CovidProvider extends Component {
  constructor(props) {
    super(props);
    this.state = {
      countries: null,
      setCountries: this.setCountries.bind(this)
    }
  }

  setCountries(countries) {
    this.setState({ countries });
  }

  render() {
    return (
      <CovidContext.Provider value={ this.state }>
        { this.props.children }
      </CovidContext.Provider>
    );
  }
}
```

Codefragment 41: Context provider bij React klasse componenten

Echter zijn wel verschillen aan te merken. Het eerste verschil is de lengte van de provider klasse die aanzienlijk is vergroot ten opzichte van de provider bij React functionele componenten. De uiteindelijke werking is wel hetzelfde. (Codefragment 41)

```
export default class Index extends Component {
  static contextType = CovidContext;

  async componentDidMount() {
    if (!this.context.countries) {
      this.context.setCountries(await getCountries());
    }
  }

  ...
```

Codefragment 42: Aanpassen van het Context-object bij React klasse component

Het uiteindelijke gebruik is ook gelijkend aan deze bij React functionele componenten, enkel de declaratie is hierbij verschillend. (Codefragment 42)

6.6.3. Vue

Naast de omslachtige manier van de Context API heeft Vue een eenvoudige manier gevonden om hetzelfde doel te bereiken.

```
mounted: async function() {
  Vue.prototype.$countries = await getCountries();
  this.isCountriesSet = true;
}
```

Codefragment 43: Globale variabele instellen bij Vue

Aan de hand van Vue.prototype.$variable is het mogelijk een variabele globaal in te stellen. Hierbij is het mogelijk hetzelfde de bereiken als React in slechts één lijn code. Het dollarteken dat voor de variabele geschreven is, heeft functioneel geen meerwaarde. Wel is het mogelijk op die manier een structuur in te brengen.
Deze variabele kan uiteindelijk aangesproken worden aan de hand van “this.$countries” in elk component. Stel dat het dollarteken niet aanwezig is, dan moet deze variabele aangesproken worden aan de hand van “this.countries”. Wanneer het component op dit

moment in het bezit is van een lokale variabele met dezelfde naam, dan bestaat de kans dat het niet mogelijk is toegang te krijgen tot de globale variabele.
Het is belangrijk deze regel in acht te nemen om dergelijke problemen te vermijden.

6.6.4. Angular

Deze functionaliteit is bij Angular te verkrijgen door gebruik te maken van services.

```
export class GlobalsService {
  private countries = null;

  constructor() { }

  setCountries(countries) {
    this.countries = countries;
  }

  getCountries() {
    return this.countries;
  }
}
```

Codefragment 44: Service bij Angular

Een service heeft dezelfde opbouw als een klasse, wat de mogelijkheid biedt functies te definiëren. In dit geval is een get()- en set()-functie voorzien. (Codefragment 44)
Het is noodzakelijk vervolgens in app.module.ts de service aan de providers-array toe te kennen. Op die manier is het mogelijk de service te importeren in de verschillende componenten.
In het component wordt vervolgens de service geïmporteerd, en wordt deze vervolgens aangemaakt in de constructor. Aan de hand van de variabele verkregen in de constructor is het mogelijk de functies te gebruiken beschreven in de specifieke service. (Codefragment 45)

```
constructor(
  private globalsService: GlobalsService,
  private covidService: CovidService,
  private populationService: PopulationService
) {
  this.countInstance = new Count();
}

ngOnInit() { }

ngDoCheck() {
  this.countries = this.globalsService.getCountries();
}
```

Codefragment 45: Aanspreken service bij Angular

6.6.5. Conclusie

Bij React functionele en klasse componenten is er niet zoveel verschil aangebracht. Het enige verschil is het aanspreken van de context. In de klasse componenten is het mogelijk om “this.context” te gebruiken. Aangezien het niet mogelijk is “this” te gebruiken binnen een functioneel component was het noodzakelijk dergelijke hook te ontwikkelen.
Op vlak van gebruiksvriendelijkheid neemt Vue de bovenhand ten opzichte van zijn medekandidaten. Deze volledige cyclus is mogelijk in slechts één lijn code, wat erg

gebruiksvriendelijk is. Hierbij is het wel noodzakelijk de regel van het dollarteken in acht te nemen.
Tot slot werkt Angular met een andere manier om dergelijke functionaliteit te bekomen, maar deze kan ook als omslachtig aanzien worden. Hierbij is het noodzakelijk een service-klasse te voorzien en deze globaal te importeren in de App module.

7. Benchmarks

Naast de vergelijking van het gebruik van hooks binnen andere frameworks is het interessant ook de benchmarks eens na te gaan. Dit is mogelijk aangezien exact hetzelfde programma is gemaakt in de verschillende frameworks.
Om de benchmarks te bepalen worden drie metingen per framework uitgevoerd. Op die manier is het mogelijk een gemiddelde waarde te berekenen. Hierbij wordt uitsluitend gekeken naar de laadtijd en de grote van het programma.
Nadien wordt een audit aan de hand van Lighthouse gegenereerd. Deze optie is reeds aanwezig in de Developer Tools van Google Chrome.

7.1. React functionele componenten

Als eerste worden de benchmarks opgemeten voor het programma dat geschreven is in React aan de hand van functionele componenten.
De resultaten zijn terug te vinden in onderstaande tabel.

Tabel 1: Benchmarks React functionele componenten

#	Laadtijd (ms)	Grote programma (MB)	Requests
1	399	2.4	11
2	344	2.4	11
3	291	2.4	11
eq	344.67	2.4	11

Voor verdere audit is het mogelijk BIJLAGE 1: BENCHMARKS REACT FUCNTIONEEL COMPONENT te raadplegen.

7.2. React klasse componenten

Als tweede worden de benchmarks opgemeten voor het programma dat geschreven is in React aan de hand van de klasse componenten.
De resultaten zijn terug te vinden in onderstaande tabel.

Tabel 2: Benchmarks React klasse componenten

#	Laadtijd (ms)	Grote programma (MB)	Requests
1	391	2.4	11
2	285	2.4	11
3	296	2.4	11
eq	324	2.4	11

Voor verdere audit is het mogelijk BIJLAGE 2: BENCHMARKS REACT KLASSE COMPONENT te raadplegen.

7.3. Vue

Als derde worden de benchmarks opgemeten voor het programma dat geschreven is in Vue.
De resultaten zijn terug te vinden in onderstaande tabel.

Tabel 3: Benchmarks Vue

#	Laadtijd (ms)	Grote programma (MB)	Requests
1	517	6.1	8
2	477	6.1	8
3	481	6.1	8
eq	491.67	6.1	8

Voor verdere audit is het mogelijk BIJLAGE 3: BENCHMARKS VUE te raadplegen.

7.4. Angular

Als vierde worden de benchmarks opgemeten voor het programma dat geschreven is in Angular.
De resultaten zijn terug te vinden in onderstaande tabel.

Tabel 4: Benchmarks Angular

#	Laadtijd (ms)	Grote programma (MB)	Requests
1	629	5.6	11
2	644	5.6	11
3	661	5.6	11
eq	644.47	5.6	11

Voor verdere audit is het mogelijk BIJLAGE 4: BENCHMARKS ANGULAR te raadplegen.

7.5. Conclusie

Uit de laadtijd van deze benchmarks is het mogelijk te concluderen dat de klasse componenten van React de beste laadtijd hanteren, gevolgd door de functionele componenten van React. De functionele componenten laden ongeveer 20ms trager dan de klasse componenten.
Het traagste framework is bijgevolg Angular met gemiddeld 644,47ms, wat ongeveer 320ms trager is dan het snelste framework. Dit betekent een dubbel zo lange laadtijd.
Ondanks de eenvoudige ontwikkeling van Vue, blijkt dat de laadsnelheid niet behoort tot de snelste. Vue bereikt met deze waarde de derde plaats. Het valt op dat de grote van het programma het grootste is bij Vue met 6.1MB, maar desondanks behoort hij niet tot het traagste framework.
Uit de audits in de bijlage gaat de hoogste score naar de functionele componenten in plaats van de klasse componenten met de snelste laadtijd. Angular en Vue scoren beide op dit rapport met een score van nul.

Conclusie

Deze bachelorproef bestaat erin een studie te voeren naar React Hooks en Context API binnen het React framework. Vervolgens wordt deze nieuwe technologie vergeleken met de hooks implementatie en State Management bij andere frameworks.

Hooks worden gebruikt om functies van klasse componenten die niet mogelijk zijn bij functionele componenten te vervangen. Op deze manier is het mogelijk een programma te schrijven bestaande enkel uit functionele componenten.

Voor de vergelijkingsapplicatie wordt een programma voorzien voor een overzicht van de COVID-19 pandemie. Dit programma is geschreven in React, Vue en Angular. In React zijn twee programma's voorzien, waarbij het eerste volledig voorzien is in klasse componenten en het tweede in functionele componenten voorzien van hooks.
Uit de vergelijking van de klasse en functionele componenten binnen het React framework is te besluiten dat de gebruiksvriendelijkheid van klasse componenten aanzienlijk hoger ligt dan deze van de functionele componenten. Hiernaast is uit de benchmarks te besluiten dat de laadtijd van klasse componenten lager is dan deze van de functionele componenten. De waarde van de gemiddelde laadtijd komt op 324ms, wat meteen ook de snelste manier is. Uit de audit uitgevoerd door Lighthouse is wel te besluiten dat de klasse componenten drie punten lager scoort dan de functionele componenten, die een score van 71 op 100 heeft. Hieruit is te besluiten dat React Hooks geen meerwaarde bieden ten opzichte van de klasse componenten. Het geeft een omslachtige en ongebruiksvriendelijke manier terug ten opzichte van de klasse componenten. De implementatie van de Context API tussen deze twee vergelijkingsprogramma's loopt nagenoeg gelijk waardoor hooks in dit geval geen eenvoudigere oplossing aanbiedt.

Wanneer de vergelijking verder wordt uitgebreid naar de andere frameworks is te besluiten dat de implementatie bij Vue het eenvoudigst is. Hierbij is het mogelijk op een snelle, eenvoudige en gebruiksvriendelijke manier dergelijke applicaties te maken. Vue heeft voor bepaalde hooks zoals de Reducer en Custom Hook, maar ook de Context API een eenvoudiger alternatief waardoor de hoeveelheid code en de complexiteit sterk gereduceerd wordt. Ondanks de goede indruk van Vue is bij de benchmarks waar te nemen dat Vue op de derde plaats staat qua laadtijd met gemiddeld 491,67ms.
Hiernaast is te besluiten dat op elke implementatie Angular het minste scoort. Dit is ook waar te nemen uit de benchmarks waarbij de laadtijd dubbel zo hoog ligt als de laagste van React klasse componenten. Voor de functionaliteit van de Reducer Hook en de Context API moet een omweg voorzien worden. Ook de hoeveelheid bestanden vermindert de overzichtelijkheid van een project in Angular.

Uit de vergelijking met andere frameworks is te besluiten dat Vue omwille van de gebruiksvriendelijkheid en de eenvoud de beste prestatie neerzet, ondanks de waarde van de laadtijd. Binnen dit framework zijn alternatieven van hooks aangebracht die voor een breder spectrum gebruikt kunnen worden. Wat momenteel niet ervaren wordt bij het gebruik van React en Angular.

Literatuurlijst

[1] „Staatsbladmonitor.be," Staatbladmonitor.be, [Online]. Available: https://www.staatsbladmonitor.be/bedrijfsfiche.html?ondernemingsnummer=0557976959. [Geopend 29 02 2020].

[2] „Cronos Groep," Cronos Groep, [Online]. Available: https://cronos-groep.be/. [Geopend 29 02 2020].

[3] „AppFoundry," AppFoundry, [Online]. Available: https://www.appfoundry.be/cases. [Geopend 29 02 2020].

[4] „Wikipedia," Wikepedia, 03 02 2020. [Online]. Available: https://nl.wikipedia.org/wiki/React. [Geopend 06 03 2020].

[5] F. Hámori, „RisingStack," RisingStack, 04 04 2018. [Online]. Available: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/. [Geopend 06 03 2020].

[6] „GitHub," WDI Lessons, 10 07 2017. [Online]. Available: https://github.com/ga-wdi-lessons/react-intro. [Geopend 05 08 2020].

[7] M. Dauwe, „GitHub," Mathias Dauwe, [Online]. Available: https://github.com/dauwem/BPHooks-Contexts. [Geopend 09 05 2020].

[8] gaearon, „GitHub," Facebook, 06 02 2019. [Online]. Available: https://github.com/facebook/react/releases?after=v16.9.0-alpha.0. [Geopend 06 03 2020].

[9] Nam, „Polidea," Polidea, [Online]. Available: https://www.polidea.com/blog/react-hooks-vs-wrapper-hell-writing-state-in-a-function-with-ease/. [Geopend 07 03 2020].

[10] R. Conf, „YouTube," 26 10 2020. [Online]. Available: https://www.youtube.com/watch?v=dpw9EHDh2bM. [Geopend 07 03 2020].

[11] D. Afonso, „Medium," Medium, 30 11 2019. [Online]. Available: https://medium.com/swlh/hooked-on-hooks-637f53ba9323. [Geopend 07 03 2020].

[12] „React," Facebook, [Online]. Available: https://reactjs.org/docs/glossary.html#lifecycle-methods. [Geopend 06 03 2020].

[13] React, „React," React, [Online]. Available: https://reactjs.org/docs/hooks-intro.html. [Geopend 07 03 2020].

[14] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-rules.html. [Geopend 03 09 2020].

[15] MDN-medewerkers, „MDN web docs," MDN web docs, 23 03 2019. [Online]. Available: https://developer.mozilla.org/nl/docs/Web/JavaScript/Reference/Operatoren/Destructuring_assignment. [Geopend 09 03 2020].

[16] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-effect.html. [Geopend 09 03 2020].

[17] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-custom.html. [Geopend 11 03 2020].

[18] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#usecontext. [Geopend 16 03 2020].

[19] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#usereducer. [Geopend 17 03 2020].

[20] „flaviocopes," flaviocopes, 18 07 2019. [Online]. Available: https://flaviocopes.com/react-hook-usecallback/. [Geopend 18 03 2020].

[21] A. Gokalp, „Hackages," Hackages, 28 03 2019. [Online]. Available: https://blog.hackages.io/react-hooks-usecallback-and-usememo-8d5bb2b67231. [Geopend 18 03 2020].

[22] A. Aziz, „DEV," DEV, 17 06 2019. [Online]. Available: https://dev.to/azrulaziz/react-usecallback-hooks-simple-explanation-1c1g. [Geopend 18 03 2020].

[23] TrinhDinhHuy, „DEV," DEV, 06 11 2019. [Online]. Available: https://dev.to/dinhhuyams/introduction-to-react-memo-usememo-and-usecallback-5ei3. [Geopend 19 03 2020].

[24] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#useref. [Geopend 19 03 2020].

[25] Bitovi, „YouTube," YouTube, 18 09 2019. [Online]. Available: https://www.youtube.com/watch?v=HxY7SzIN44o. [Geopend 20 03 2020].

[26] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#useimperativehandle. [Geopend 20 03 2020].

[27] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#uselayouteffect. [Geopend 20 03 2020].

[28] „React," React, [Online]. Available: https://reactjs.org/docs/hooks-reference.html#usedebugvalue. [Geopend 20 03 2020].

[29] B. Ziroll, „freeCodeCamp," freeCodeCamp, 08 11 2019. [Online]. Available: https://www.freecodecamp.org/news/react-context-in-5-minutes/. [Geopend 14 03 2020].

[30] W. Bos, „wesbos," wesbos, 13 03 2018. [Online]. Available: https://wesbos.com/react-context/. [Geopend 14 03 2020].

[31] B. Pak, „DEV," DEV, 04 05 2019. [Online]. Available: https://dev.to/bouhm/thinking-in-react-component-composition-fp5. [Geopend 13 03 2020].

[32] „React," React, [Online]. Available: https://reactjs.org/docs/context.html. [Geopend 14 03 2020].

[33] „React," React, [Online]. Available: https://reactjs.org/docs/composition-vs-inheritance.html. [Geopend 14 03 2020].

[34] „HotFrameworks," HotFrameworks, [Online]. Available: https://hotframeworks.com/. [Geopend 23 03 2020].

[35] M. Dauwe, „GitHub," Mathias Dauwe, [Online]. Available: https://github.com/dauwem/BPCompFrameworks-Hooks. [Geopend 09 05 2020].

[36] „Vue," Vue, [Online]. Available: https://vuejs.org/v2/guide/syntax.html. [Geopend 25 04 2020].

[37] W. Maj, „Github," Wojciech Maj, 23 03 2020. [Online]. Available: https://github.com/wojtekmaj/react-lifecycle-methods-diagram. [Geopend 25 04 2020].

[38] „Vue," Vue, [Online]. Available: https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks. [Geopend 25 04 2020].

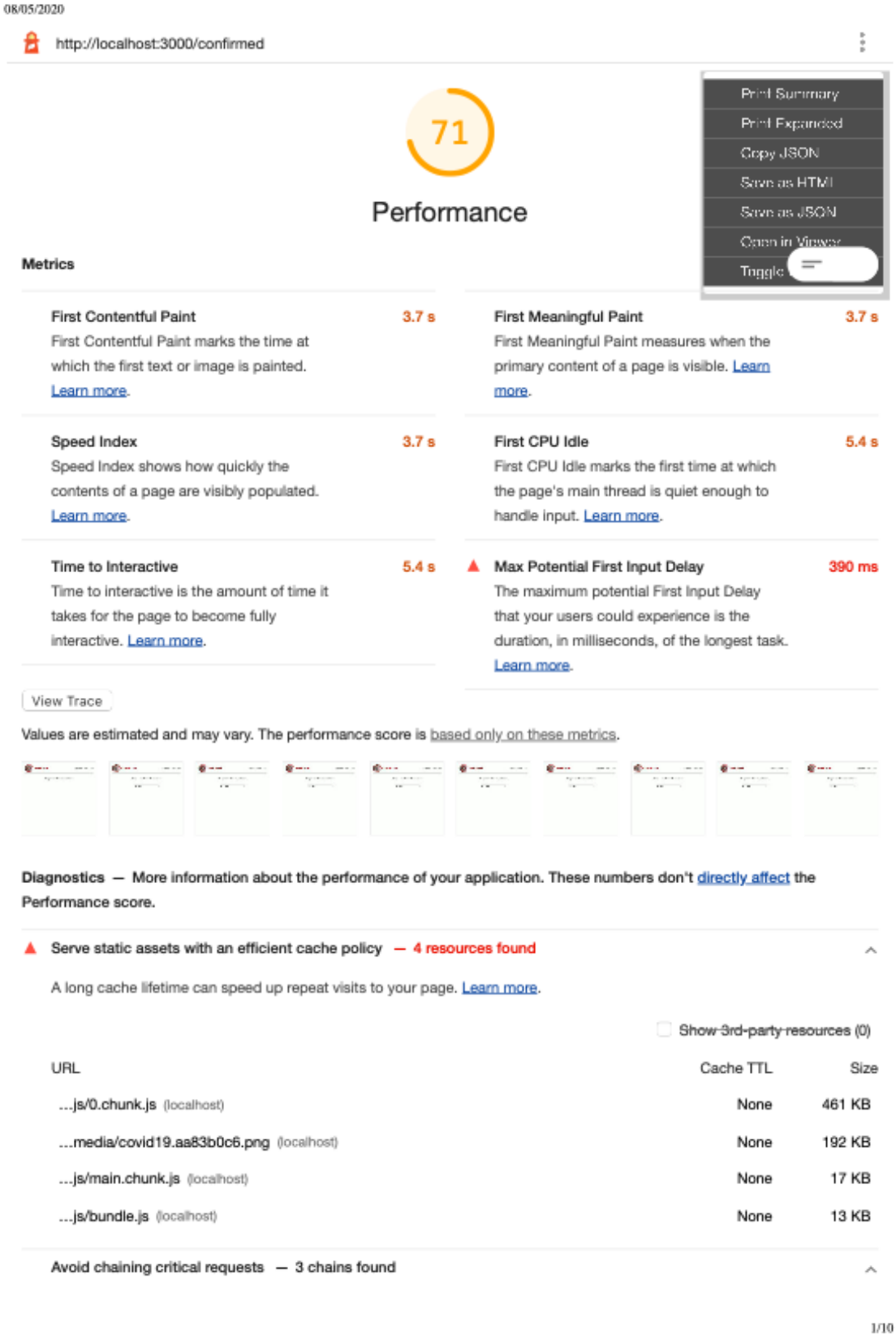
[39] S. Debasis, „DZone," DZone, 25 07 2018. [Online]. Available: https://dzone.com/articles/angular-6-part-3-life-cycle-of-a-component. [Geopend 02 05 2020].

[40] „Vue," Vue, [Online]. Available: https://vuejs.org/v2/guide/mixins.html. [Geopend 04 05 2020].

[41] „Wikipedia," Wikipedia, 02 05 2020. [Online]. Available: https://nl.wikipedia.org/wiki/Application_programming_interface. [Geopend 06 05 2020].

Bijlagenoverzicht

BIJLAGE 1: BENCHMARKS REACT FUCNTIONEEL COMPONENT



08/05/2020

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more.

Maximum critical path latency: **280 ms**

Initial Navigation

/confirmed (localhost)

...js/bundle.js (localhost) - **10 ms, 12.8 KB**

...js/0.chunk.js (localhost) - **100 ms, 461.17 KB**

...js/main.chunk.js (localhost) - **10 ms, 16.55 KB**

/manifest.json (localhost) - **0 ms, 0.8 KB**

User Timing marks and measures — 128 user timings

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

Use the React DevTools Profiler, which makes use of the Profiler API, to measure the rendering performance of your components. Learn more.

Name	Type	Start Time	Duration
(React Tree Reconciliation: Completed Root)	Measure	221.72 ms	15.65 ms
CovidProvider [mount]	Measure	223.57 ms	13.62 ms
Index [mount]	Measure	224.35 ms	12.82 ms
Header [mount]	Measure	225.52 ms	7.36 ms
Link [mount]	Measure	226.16 ms	4.58 ms
Location [mount]	Measure	226.48 ms	4.22 ms
LocationProvider [mount]	Measure	226.65 ms	3.96 ms
Link [mount]	Measure	230.95 ms	0.52 ms
Location [mount]	Measure	231.03 ms	0.41 ms
LocationProvider [mount]	Measure	231.1 ms	0.31 ms
Link [mount]	Measure	231.61 ms	0.48 ms
Location [mount]	Measure	231.69 ms	0.37 ms
LocationProvider [mount]	Measure	231.78 ms	0.25 ms
Link [mount]	Measure	232.17 ms	0.43 ms
Location [mount]	Measure	232.26 ms	0.32 ms
LocationProvider [mount]	Measure	232.33 ms	0.23 ms

2/10

08/05/2020

Name	Type	Start Time	Duration
MainContainer [mount]	Measure	232.9 ms	4.24 ms
Router [mount]	Measure	233.08 ms	3.98 ms
Location [mount]	Measure	233.23 ms	3.82 ms
LocationProvider [mount]	Measure	233.29 ms	3.74 ms
RouterImpl [mount]	Measure	233.45 ms	3.56 ms
FocusHandler [mount]	Measure	234.97 ms	2.02 ms
FocusHandlerImpl [mount]	Measure	235.15 ms	1.81 ms
ConfirmedGraphContainer [mount]	Measure	235.65 ms	0.99 ms
Title02 [mount]	Measure	236.33 ms	0.14 ms
MainDropdown [mount]	Measure	236.51 ms	0.1 ms
(Committing Changes)	Measure	237.86 ms	3.03 ms
(Committing Snapshot Effects: 0 Total)	Measure	237.95 ms	1.12 ms
(Committing Host Effects: 10 Total)	Measure	239.12 ms	0.65 ms
(Calling Lifecycle Methods: 9 Total)	Measure	239.89 ms	0.94 ms
LocationProvider.componentDidMount	Measure	240.32 ms	0.13 ms
FocusHandlerImpl.componentDidMount	Measure	240.66 ms	0.07 ms
(React Tree Reconciliation: Completed Root)	Measure	294.95 ms	19.41 ms
CovidProvider [update]	Measure	295.14 ms	19.2 ms
Index [update]	Measure	295.77 ms	18.54 ms
Header [update]	Measure	296.23 ms	2.38 ms
Link [update]	Measure	296.4 ms	0.88 ms
Location [update]	Measure	296.48 ms	0.78 ms
LocationProvider [update]	Measure	296.54 ms	0.68 ms
Link [update]	Measure	297.37 ms	0.29 ms
Location [update]	Measure	297.43 ms	0.22 ms
LocationProvider [update]	Measure	297.49 ms	0.13 ms
Link [update]	Measure	297.71 ms	0.27 ms
Location [update]	Measure	297.77 ms	0.19 ms

3/10

08/05/2020

Name	Type	Start Time	Duration
LocationProvider [update]	Measure	297.82 ms	0.12 ms
Link [update]	Measure	298.02 ms	0.5 ms
Location [update]	Measure	298.1 ms	0.42 ms
LocationProvider [update]	Measure	298.14 ms	0.33 ms
MainContainer [update]	Measure	298.65 ms	15.63 ms
Router [update]	Measure	298.76 ms	15.48 ms
Location [update]	Measure	298.83 ms	15.38 ms
LocationProvider [update]	Measure	298.89 ms	15.31 ms
RouterImpl [update]	Measure	298.96 ms	15.21 ms
FocusHandler [update]	Measure	299.32 ms	14.83 ms
FocusHandlerImpl [update]	Measure	299.39 ms	14.72 ms
ConfirmedGraphContainer [update]	Measure	299.58 ms	14.44 ms
Title02 [update]	Measure	300.02 ms	0.14 ms
MainDropdown [update]	Measure	300.21 ms	13.78 ms
(Committing Changes)	Measure	314.39 ms	1.63 ms
(Committing Snapshot Effects: 0 Total)	Measure	314.4 ms	0.37 ms
(Committing Host Effects: 13 Total)	Measure	314.79 ms	0.71 ms
(Calling Lifecycle Methods: 12 Total)	Measure	315.52 ms	0.47 ms
LocationProvider.componentDidUpdate	Measure	315.74 ms	0.07 ms
FocusHandlerImpl.componentDidUpdate	Measure	315.85 ms	0.07 ms
(React Tree Reconciliation)	Mark	221.73 ms	
CovidProvider [mount] (#4)	Mark	223.57 ms	
Index [mount] (#8)	Mark	224.35 ms	
Header [mount] (#10)	Mark	225.52 ms	
Link [mount] (#18)	Mark	226.16 ms	
Location [mount] (#22)	Mark	226.49 ms	
LocationProvider [mount] (#26)	Mark	226.65 ms	
Link [mount] (#39)	Mark	230.95 ms	

4/10

08/05/2020

Name	Type	Start Time	Duration
Location [mount] (#43)	Mark	231.03 ms	
LocationProvider [mount] (#47)	Mark	231.1 ms	
Link [mount] (#54)	Mark	231.62 ms	
Location [mount] (#58)	Mark	231.69 ms	
LocationProvider [mount] (#62)	Mark	231.78 ms	
Link [mount] (#69)	Mark	232.18 ms	
Location [mount] (#73)	Mark	232.26 ms	
LocationProvider [mount] (#77)	Mark	232.33 ms	
MainContainer [mount] (#11)	Mark	232.91 ms	
Router [mount] (#86)	Mark	233.08 ms	
Location [mount] (#90)	Mark	233.23 ms	
LocationProvider [mount] (#94)	Mark	233.29 ms	
RouterImpl [mount] (#98)	Mark	233.45 ms	
FocusHandler [mount] (#102)	Mark	234.98 ms	
FocusHandlerImpl [mount] (#106)	Mark	235.15 ms	
ConfirmedGraphContainer [mount] (#112)	Mark	235.66 ms	
Title02 [mount] (#114)	Mark	236.33 ms	
MainDropdown [mount] (#115)	Mark	236.51 ms	
(Committing Changes)	Mark	237.86 ms	
(Committing Snapshot Effects)	Mark	237.95 ms	
(Committing Host Effects)	Mark	239.12 ms	
(Calling Lifecycle Methods)	Mark	239.89 ms	
LocationProvider.componentDidMount (#26)	Mark	240.32 ms	
FocusHandlerImpl.componentDidMount (#106)	Mark	240.66 ms	
(React Tree Reconciliation)	Mark	294.95 ms	
CovidProvider [update] (#4)	Mark	295.14 ms	
Index [update] (#8)	Mark	295.78 ms	
Header [update] (#10)	Mark	296.24 ms	

5/10

08/05/2020

Name	Type	Start Time	Duration
Link [update] (#18)	Mark	296.4 ms	
Location [update] (#22)	Mark	296.48 ms	
LocationProvider [update] (#26)	Mark	296.54 ms	
Link [update] (#39)	Mark	297.37 ms	
Location [update] (#43)	Mark	297.43 ms	
LocationProvider [update] (#47)	Mark	297.49 ms	
Link [update] (#54)	Mark	297.71 ms	
Location [update] (#58)	Mark	297.77 ms	
LocationProvider [update] (#62)	Mark	297.82 ms	
Link [update] (#69)	Mark	298.03 ms	
Location [update] (#73)	Mark	298.1 ms	
LocationProvider [update] (#77)	Mark	298.15 ms	
MainContainer [update] (#11)	Mark	298.66 ms	
Router [update] (#86)	Mark	298.77 ms	
Location [update] (#90)	Mark	298.83 ms	
LocationProvider [update] (#94)	Mark	298.89 ms	
RouterImpl [update] (#98)	Mark	298.97 ms	
FocusHandler [update] (#102)	Mark	299.33 ms	
FocusHandlerImpl [update] (#106)	Mark	299.4 ms	
ConfirmedGraphContainer [update] (#112)	Mark	299.59 ms	
Title02 [update] (#114)	Mark	300.03 ms	
MainDropdown [update] (#115)	Mark	300.22 ms	
(Committing Changes)	Mark	314.39 ms	
(Committing Snapshot Effects)	Mark	314.41 ms	
(Committing Host Effects)	Mark	314.79 ms	
(Calling Lifecycle Methods)	Mark	315.53 ms	
LocationProvider.componentDidUpdate (#26)	Mark	315.74 ms	
FocusHandlerImpl.componentDidUpdate (#106)	Mark	315.85 ms	

6/10

08/05/2020

Keep request counts low and transfer sizes small — 9 requests • 851 KB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	9	851 KB
Script	4	640 KB
Image	1	192 KB
Other	3	17 KB
Document	1	1 KB
Stylesheet	0	0 KB
Media	0	0 KB
Font	0	0 KB
Third-party	2	159 KB

Passed audits (20)

Eliminate render-blocking resources

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more.

Properly size images

Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

Defer offscreen images

Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

Minify CSS

Minifying CSS files can reduce network payload sizes. Learn more.

If your build system minifies your CSS files automatically, ensure that you are deploying the production build of your application. You can check this with the React Developer Tools extension. Learn more.

Minify JavaScript

Minifying JavaScript files can reduce payload sizes and script parse time. Learn more.

If your build system minifies your JS files automatically, ensure that you are deploying the production build of your application. You can check this with the React Developer Tools extension. Learn more.

Remove unused CSS

Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more.

Efficiently encode images

7/10

08/05/2020

Optimized images load faster and consume less cellular data. Learn more.

Serve images in next-gen formats

Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

Enable text compression

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

Preconnect to required origins

Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn more.

Server response times are low (TTFB) — **Root document took 0 ms**

Time To First Byte identifies the time at which your server sends a response. Learn more.

If you are server-side rendering any React components, consider using `renderToNodeStream()` or `renderToStaticNodeStream()` to allow the client to receive and hydrate different parts of the markup instead of all at once. Learn more.

Avoid multiple page redirects

Redirects introduce additional delays before the page can be loaded. Learn more.

If you are using React Router, minimize usage of the `<Redirect>` component for route navigations.

Preload key requests

Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more.

Use video formats for animated content

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

Avoids enormous network payloads — **Total size was 851 KB**

Large network payloads cost users real money and are highly correlated with long load times. Learn more.

☑ Show 3rd-party resources (1)

URL	Size
...js/0.chunk.js (localhost)	461 KB
...media/covid19.aa83b0c6.png (localhost)	192 KB
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/react_devtools_backend.js	149 KB
...js/main.chunk.js (localhost)	17 KB
...js/bundle.js (localhost)	13 KB
/countries (api.covid19api.com)	9 KB

8/10

08/05/2020

URL	Size
/logo192.png (localhost)	7 KB
/confirmed (localhost)	1 KB
/manifest.json (localhost)	1 KB

Avoids an excessive DOM size — 269 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

Consider using a "windowing" library like `react-window` to minimize the number of DOM nodes created if you are rendering many repeated elements on the page. Learn more. Also, minimize unecessary re-renders using shouldComponentUpdate, PureComponent, or React.memo and skip effects only until certain dependencies have changed if you are using the Effect hook to improve runtime performance.

Statistic	Element	Value
Total DOM Elements		269
Maximum DOM Depth	`<a aria-current="page" href="/confirmed">`	7
Maximum Child Elements	`<select class="MainDropdown">`	249

JavaScript execution time — 0.6 s

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

Show 3rd-party resources (0)

URL	Total CPU Time	Script Evaluation	Script Parse
Other	560 ms	110 ms	3 ms
...js/main.chunk.js (localhost)	180 ms	175 ms	5 ms
...js/0.chunk.js (localhost)	141 ms	30 ms	111 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/react_devtools_backend.js	66 ms	46 ms	20 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/injectGlobalHook.js	60 ms	31 ms	28 ms

Minimizes main-thread work — 1.0 s

Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this. Learn more

Category	Time Spent
Script Evaluation	418 ms
Other	351 ms
Script Parsing & Compilation	180 ms
Parse HTML & CSS	62 ms

9/10

08/05/2020

Category	Time Spent
Style & Layout	14 ms
Garbage Collection	12 ms
Rendering	8 ms

All text remains visible during webfont loads

Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

Minimize third-party usage

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Learn more.

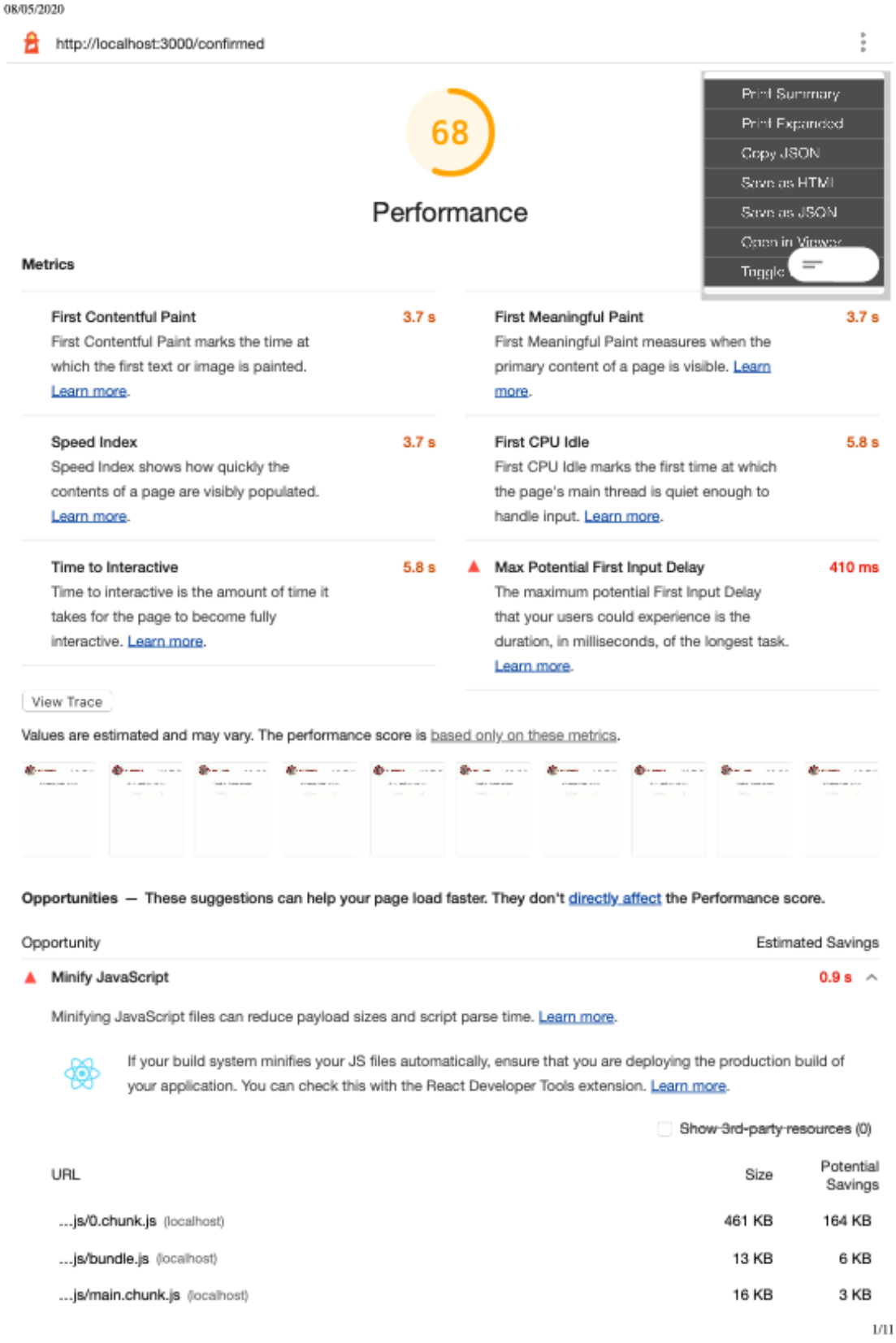
Runtime Settings

URL	http://localhost:3000/confirmed
Fetch time	May 8, 2020, 2:57 PM GMT+2
Device	Emulated Desktop
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1408

Generated by **Lighthouse** 5.7.0 | File an issue

10/10

BIJLAGE 2: BENCHMARKS REACT KLASSE COMPONENT



08/05/2020

Serve images in next-gen formats 0.79 s

Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

Show 3rd-party resources (0)

URL	Size	Potential Savings
 ...media/covid19.aa83b0c6.png (localhost)	192 KB	170 KB

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

▲ **Serve static assets with an efficient cache policy** — 4 resources found

A long cache lifetime can speed up repeat visits to your page. Learn more.

Show 3rd-party resources (0)

URL	Cache TTL	Size
...js/0.chunk.js (localhost)	None	461 KB
...media/covid19.aa83b0c6.png (localhost)	None	192 KB
...js/main.chunk.js (localhost)	None	16 KB
...js/bundle.js (localhost)	None	13 KB

Avoid chaining critical requests — 3 chains found

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more.

Maximum critical path latency: **270 ms**

Initial Navigation

/confirmed (localhost)

...js/bundle.js (localhost) - **20 ms, 12.8 KB**

...js/0.chunk.js (localhost) - **100 ms, 461.17 KB**

...js/main.chunk.js (localhost) - **20 ms, 16.45 KB**

/manifest.json (localhost) - **10 ms, 0.8 KB**

User Timing marks and measures — 132 user timings

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

Use the React DevTools Profiler, which makes use of the Profiler API, to measure the rendering performance of your components. Learn more.

2/11

08/05/2020

Name	Type	Start Time	Duration
(React Tree Reconciliation: Completed Root)	Measure	209.34 ms	15.64 ms
CovidProvider [mount]	Measure	211.15 ms	13.64 ms
Index [mount]	Measure	212.13 ms	12.64 ms
Header [mount]	Measure	213.21 ms	7.37 ms
Link [mount]	Measure	213.85 ms	4.68 ms
Location [mount]	Measure	214.24 ms	4.24 ms
LocationProvider [mount]	Measure	214.57 ms	3.71 ms
Link [mount]	Measure	218.81 ms	0.59 ms
Location [mount]	Measure	218.92 ms	0.46 ms
LocationProvider [mount]	Measure	218.99 ms	0.36 ms
Link [mount]	Measure	219.54 ms	0.38 ms
Location [mount]	Measure	219.61 ms	0.28 ms
LocationProvider [mount]	Measure	219.67 ms	0.19 ms
Link [mount]	Measure	219.98 ms	0.37 ms
Location [mount]	Measure	220.06 ms	0.27 ms
LocationProvider [mount]	Measure	220.12 ms	0.18 ms
MainContainer [mount]	Measure	220.61 ms	4.13 ms
Router [mount]	Measure	220.85 ms	3.83 ms
Location [mount]	Measure	221.01 ms	3.64 ms
LocationProvider [mount]	Measure	221.09 ms	3.55 ms
RouterImpl [mount]	Measure	221.25 ms	3.37 ms
FocusHandler [mount]	Measure	222.77 ms	1.82 ms
FocusHandlerImpl [mount]	Measure	222.94 ms	1.62 ms
ConfirmedGraphContainer [mount]	Measure	223.46 ms	0.8 ms
Title02 [mount]	Measure	223.82 ms	0.22 ms
MainDropdown [mount]	Measure	224.08 ms	0.14 ms
(Committing Changes)	Measure	225.48 ms	3.46 ms
(Committing Snapshot Effects: 0 Total)	Measure	225.57 ms	0.76 ms

3/11

08/05/2020

Name	Type	Start Time	Duration
(Committing Host Effects: 9 Total)	Measure	226.39 ms	0.63 ms
(Calling Lifecycle Methods: 8 Total)	Measure	227.13 ms	1.74 ms
LocationProvider.componentDidMount	Measure	227.57 ms	0.13 ms
FocusHandlerImpl.componentDidMount	Measure	227.86 ms	0.07 ms
Index.componentDidMount	Measure	227.96 ms	0.8 ms
(React Tree Reconciliation: Completed Root)	Measure	278.83 ms	20.2 ms
CovidProvider [update]	Measure	279.04 ms	19.97 ms
Index [update]	Measure	279.65 ms	19.32 ms
Header [update]	Measure	280.02 ms	2.38 ms
Link [update]	Measure	280.24 ms	0.78 ms
Location [update]	Measure	280.32 ms	0.68 ms
LocationProvider [update]	Measure	280.47 ms	0.49 ms
Link [update]	Measure	281.11 ms	0.41 ms
Location [update]	Measure	281.22 ms	0.27 ms
LocationProvider [update]	Measure	281.32 ms	0.14 ms
Link [update]	Measure	281.57 ms	0.32 ms
Location [update]	Measure	281.63 ms	0.23 ms
LocationProvider [update]	Measure	281.69 ms	0.16 ms
Link [update]	Measure	281.93 ms	0.36 ms
Location [update]	Measure	281.99 ms	0.26 ms
LocationProvider [update]	Measure	282.05 ms	0.13 ms
MainContainer [update]	Measure	282.44 ms	16.51 ms
Router [update]	Measure	282.53 ms	16.38 ms
Location [update]	Measure	283.09 ms	15.8 ms
LocationProvider [update]	Measure	283.18 ms	15.69 ms
RouterImpl [update]	Measure	283.26 ms	15.59 ms
FocusHandler [update]	Measure	284.4 ms	14.42 ms
FocusHandlerImpl [update]	Measure	284.51 ms	14.28 ms

4/11

08/05/2020

Name	Type	Start Time	Duration
ConfirmedGraphContainer [update]	Measure	284.65 ms	14.04 ms
Title02 [update]	Measure	284.77 ms	0.11 ms
MainDropdown [update]	Measure	284.91 ms	13.73 ms
(Committing Changes)	Measure	299.05 ms	2.78 ms
(Committing Snapshot Effects: 0 Total)	Measure	299.07 ms	0.36 ms
(Committing Host Effects: 13 Total)	Measure	299.44 ms	1.18 ms
(Calling Lifecycle Methods: 12 Total)	Measure	300.65 ms	1.14 ms
LocationProvider.componentDidUpdate	Measure	300.93 ms	0.08 ms
ConfirmedGraphContainer.componentDidUpdate	Measure	301.08 ms	0.13 ms
FocusHandlerImpl.componentDidUpdate	Measure	301.26 ms	0.13 ms
(React Tree Reconciliation)	Mark	209.35 ms	
CovidProvider [mount] (#4)	Mark	211.16 ms	
Index [mount] (#8)	Mark	212.14 ms	
Header [mount] (#10)	Mark	213.22 ms	
Link [mount] (#18)	Mark	213.86 ms	
Location [mount] (#22)	Mark	214.25 ms	
LocationProvider [mount] (#26)	Mark	214.58 ms	
Link [mount] (#39)	Mark	218.82 ms	
Location [mount] (#43)	Mark	218.92 ms	
LocationProvider [mount] (#47)	Mark	219 ms	
Link [mount] (#54)	Mark	219.54 ms	
Location [mount] (#58)	Mark	219.61 ms	
LocationProvider [mount] (#62)	Mark	219.68 ms	
Link [mount] (#69)	Mark	219.99 ms	
Location [mount] (#73)	Mark	220.06 ms	
LocationProvider [mount] (#77)	Mark	220.13 ms	
MainContainer [mount] (#11)	Mark	220.62 ms	
Router [mount] (#86)	Mark	220.85 ms	

5/11

08/05/2020

Name	Type	Start Time	Duration
Location [mount] (#90)	Mark	221.02 ms	
LocationProvider [mount] (#94)	Mark	221.1 ms	
RouterImpl [mount] (#98)	Mark	221.25 ms	
FocusHandler [mount] (#102)	Mark	222.78 ms	
FocusHandlerImpl [mount] (#106)	Mark	222.95 ms	
ConfirmedGraphContainer [mount] (#112)	Mark	223.46 ms	
Title02 [mount] (#114)	Mark	223.83 ms	
MainDropdown [mount] (#115)	Mark	224.08 ms	
(Committing Changes)	Mark	225.49 ms	
(Committing Snapshot Effects)	Mark	225.58 ms	
(Committing Host Effects)	Mark	226.39 ms	
(Calling Lifecycle Methods)	Mark	227.14 ms	
LocationProvider.componentDidMount (#26)	Mark	227.57 ms	
FocusHandlerImpl.componentDidMount (#106)	Mark	227.86 ms	
Index.componentDidMount (#8)	Mark	227.97 ms	
(React Tree Reconciliation)	Mark	278.84 ms	
CovidProvider [update] (#4)	Mark	279.04 ms	
Index [update] (#8)	Mark	279.67 ms	
Header [update] (#10)	Mark	280.03 ms	
Link [update] (#18)	Mark	280.25 ms	
Location [update] (#22)	Mark	280.33 ms	
LocationProvider [update] (#26)	Mark	280.48 ms	
Link [update] (#39)	Mark	281.12 ms	
Location [update] (#43)	Mark	281.23 ms	
LocationProvider [update] (#47)	Mark	281.32 ms	
Link [update] (#54)	Mark	281.58 ms	
Location [update] (#58)	Mark	281.64 ms	
LocationProvider [update] (#62)	Mark	281.69 ms	

6/11

08/05/2020

Name	Type	Start Time	Duration
Link [update] (#69)	Mark	281.94 ms	
Location [update] (#73)	Mark	282 ms	
LocationProvider [update] (#77)	Mark	282.05 ms	
MainContainer [update] (#11)	Mark	282.44 ms	
Router [update] (#86)	Mark	282.54 ms	
Location [update] (#90)	Mark	283.1 ms	
LocationProvider [update] (#94)	Mark	283.18 ms	
RouterImpl [update] (#98)	Mark	283.26 ms	
FocusHandler [update] (#102)	Mark	284.42 ms	
FocusHandlerImpl [update] (#106)	Mark	284.52 ms	
ConfirmedGraphContainer [update] (#112)	Mark	284.67 ms	
Title02 [update] (#114)	Mark	284.78 ms	
MainDropdown [update] (#115)	Mark	284.91 ms	
(Committing Changes)	Mark	299.06 ms	
(Committing Snapshot Effects)	Mark	299.08 ms	
(Committing Host Effects)	Mark	299.45 ms	
(Calling Lifecycle Methods)	Mark	300.67 ms	
LocationProvider.componentDidUpdate (#26)	Mark	300.94 ms	
ConfirmedGraphContainer.componentDidUpdate (#112)	Mark	301.11 ms	
FocusHandlerImpl.componentDidUpdate (#106)	Mark	301.26 ms	

Keep request counts low and transfer sizes small — 9 requests • 850 KB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	9	850 KB
Script	4	640 KB
Image	1	192 KB
Other	3	17 KB
Document	1	1 KB

7/11

08/05/2020

Resource Type	Requests	Transfer Size
Stylesheet	0	0 KB
Media	0	0 KB
Font	0	0 KB
Third-party	2	154 KB

Passed audits (18)

Eliminate render-blocking resources

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more.

Properly size images

Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

Defer offscreen images

Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

Minify CSS

Minifying CSS files can reduce network payload sizes. Learn more.

If your build system minifies your CSS files automatically, ensure that you are deploying the production build of your application. You can check this with the React Developer Tools extension. Learn more.

Remove unused CSS

Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more.

Efficiently encode images

Optimized images load faster and consume less cellular data. Learn more.

Enable text compression

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

Preconnect to required origins

Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn more.

Server response times are low (TTFB) — Root document took 0 ms

Time To First Byte identifies the time at which your server sends a response. Learn more.

If you are server-side rendering any React components, consider using `renderToNodeStream()` or `renderToStaticNodeStream()` to allow the client to receive and hydrate different parts of the markup instead of all at once. Learn more.

8/11

08/05/2020

Avoid multiple page redirects

Redirects introduce additional delays before the page can be loaded. Learn more.

If you are using React Router, minimize usage of the `<Redirect>` component for route navigations.

Preload key requests

Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more.

Use video formats for animated content

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

Avoids enormous network payloads — Total size was 850 KB

Large network payloads cost users real money and are highly correlated with long load times. Learn more.

Show 3rd-party resources (1)

URL	Size
...js/0.chunk.js (localhost)	461 KB
...media/covid19.aa83b0c6.png (localhost)	192 KB
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/react_devtools_backend.js	149 KB
...js/main.chunk.js (localhost)	16 KB
...js/bundle.js (localhost)	13 KB
/logo192.png (localhost)	11 KB
/countries (api.covid19api.com)	5 KB
/confirmed (localhost)	1 KB
/manifest.json (localhost)	1 KB

Avoids an excessive DOM size — 269 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

Consider using a "windowing" library like `react-window` to minimize the number of DOM nodes created if you are rendering many repeated elements on the page. Learn more. Also, minimize unecessary re-renders using shouldComponentUpdate, PureComponent, or React.memo and skip effects only until certain dependencies have changed if you are using the Effect hook to improve runtime performance.

Statistic	Element	Value
Total DOM Elements		269
Maximum DOM Depth	`<a aria-current="page" href="/confirmed">`	7
Maximum Child Elements	`<select class="MainDropdown">`	249

JavaScript execution time — 0.6 s

9/11

08/05/2020

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

Show 3rd-party resources (0)

URL	Total CPU Time	Script Evaluation	Script Parse
Other	495 ms	108 ms	2 ms
...js/main.chunk.js (localhost)	186 ms	180 ms	5 ms
...js/0.chunk.js (localhost)	133 ms	19 ms	113 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/injectGlobalHook.js	89 ms	73 ms	15 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/react_devtools_backend.js	67 ms	47 ms	20 ms

Minimizes main-thread work — 1.0 s

Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this. Learn more

Category	Time Spent
Script Evaluation	460 ms
Other	298 ms
Script Parsing & Compilation	169 ms
Parse HTML & CSS	51 ms
Garbage Collection	15 ms
Style & Layout	14 ms
Rendering	6 ms

All text remains visible during webfont loads

Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

Minimize third-party usage

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Learn more.

Runtime Settings

URL	http://localhost:3000/confirmed
Fetch time	May 8, 2020, 3:14 PM GMT+2
Device	Emulated Desktop

10/11

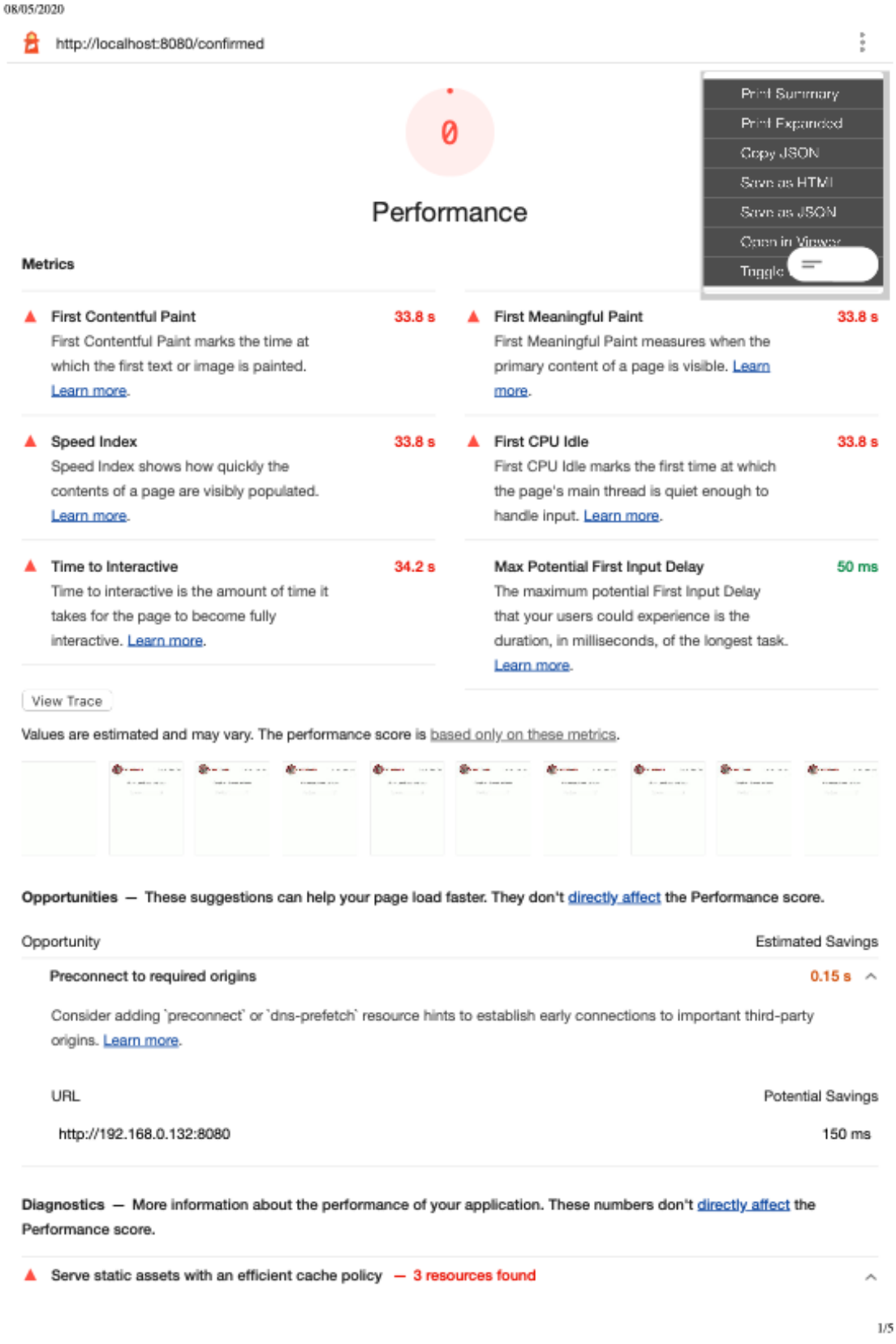
08/05/2020

Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1490

Generated by **Lighthouse** 5.7.0 | File an issue

11/11

BIJLAGE 3: BENCHMARKS VUE



08/05/2020

A long cache lifetime can speed up repeat visits to your page. Learn more.

Show 3rd-party resources (0)

URL	Cache TTL	Size
/js/chunk-vendors.js (localhost)	None	5,358 KB
/js/app.js (localhost)	None	985 KB
...images/covid19.png (localhost)	None	192 KB

Avoid enormous network payloads — Total size was 6,545 KB

Large network payloads cost users real money and are highly correlated with long load times. Learn more.

Show 3rd-party resources (2)

URL	Size
/js/chunk-vendors.js (localhost)	5,358 KB
/js/app.js (localhost)	985 KB
...images/covid19.png (localhost)	192 KB
/countries (api.covid19api.com)	9 KB
/confirmed (localhost)	1 KB
/sockjs-node/info?t=158... (192.168.0.132)	0 KB

Keep request counts low and transfer sizes small — 6 requests • 6,545 KB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	6	6,545 KB
Script	2	6,342 KB
Image	1	192 KB
Other	2	10 KB
Document	1	1 KB
Stylesheet	0	0 KB
Media	0	0 KB
Font	0	0 KB
Third-party	2	10 KB

Passed audits (20)

Eliminate render-blocking resources

2/5

08/05/2020

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more.

Properly size images

Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

Defer offscreen images

Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

Minify CSS

Minifying CSS files can reduce network payload sizes. Learn more.

Minify JavaScript

Minifying JavaScript files can reduce payload sizes and script parse time. Learn more.

Remove unused CSS

Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more.

Efficiently encode images

Optimized images load faster and consume less cellular data. Learn more.

Serve images in next-gen formats

Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

Enable text compression

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

Server response times are low (TTFB) — Root document took 0 ms

Time To First Byte identifies the time at which your server sends a response. Learn more.

Avoid multiple page redirects

Redirects introduce additional delays before the page can be loaded. Learn more.

Preload key requests

Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more.

Use video formats for animated content

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

Avoids an excessive DOM size — 269 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

3/5

08/05/2020

Statistic	Element	Value
Total DOM Elements		269
Maximum DOM Depth	`<a href="/confirmed" class="router-link-exact-active router-link-active">`	7
Maximum Child Elements	`<select class="MainDropdown">`	249

Avoid chaining critical requests

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more.

Maximum critical path latency: **0 ms**

Initial Navigation

/confirmed (localhost) - **0 ms, 0.94 KB**

User Timing marks and measures

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

JavaScript execution time — 1.1 s

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

Show 3rd-party resources (0)

URL	Total CPU Time	Script Evaluation	Script Parse
/js/app.js (localhost)	677 ms	655 ms	21 ms
Other	540 ms	62 ms	3 ms
/js/chunk-vendors.js (localhost)	284 ms	2 ms	283 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/injectGlobalHook.js	60 ms	44 ms	16 ms

Minimizes main-thread work — 1.6 s

Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this. Learn more

Category	Time Spent
Script Evaluation	806 ms
Other	390 ms
Script Parsing & Compilation	331 ms
Parse HTML & CSS	45 ms
Garbage Collection	21 ms
Rendering	11 ms

4/5

08/05/2020

Category	Time Spent
Style & Layout	9 ms

All text remains visible during webfont loads

Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

Minimize third-party usage

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Learn more.

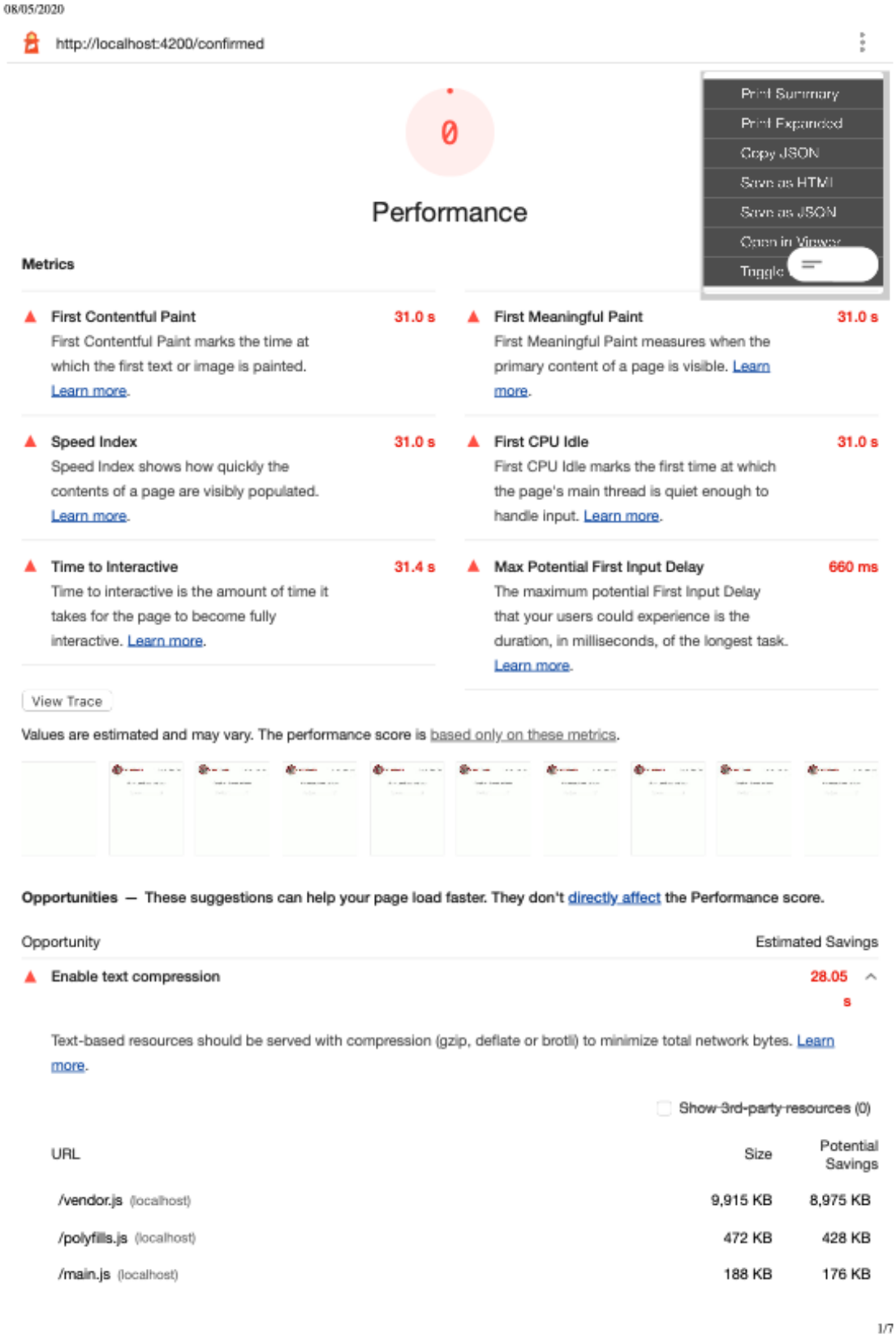
Runtime Settings

URL	http://localhost:8080/confirmed
Fetch time	May 8, 2020, 3:12 PM GMT+2
Device	Emulated Desktop
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1390

Generated by **Lighthouse** 5.7.0 | File an issue

5/5

BIJLAGE 4: BENCHMARKS ANGULAR



08/05/2020

http://localhost:4200/confirmed

0

Performance

Print Summary
Print Expanded
Copy JSON
Save as HTML
Save as JSON
Open in Viewer
Toggle

Metrics

First Contentful Paint 31.0 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.

First Meaningful Paint 31.0 s
First Meaningful Paint measures when the primary content of a page is visible. Learn more.

Speed Index 31.0 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.

First CPU Idle 31.0 s
First CPU Idle marks the first time at which the page's main thread is quiet enough to handle input. Learn more.

Time to Interactive 31.4 s
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.

Max Potential First Input Delay 660 ms
The maximum potential First Input Delay that your users could experience is the duration, in milliseconds, of the longest task. Learn more.

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Enable text compression	28.05 s

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

Show 3rd-party resources (0)

URL	Size	Potential Savings
/vendor.js (localhost)	9,915 KB	8,975 KB
/polyfills.js (localhost)	472 KB	428 KB
/main.js (localhost)	188 KB	176 KB

1/7

08/05/2020

URL	Size	Potential Savings
/styles.js (localhost)	34 KB	29 KB
/runtime.js (localhost)	12 KB	11 KB

▲ **Minify JavaScript** 11.1 s

Minifying JavaScript files can reduce payload sizes and script parse time. Learn more.

If you are using Angular CLI, ensure that builds are generated in production mode. Learn more.

Show 3rd-party resources (0)

URL	Size	Potential Savings
/vendor.js (localhost)	4,958 KB	1,987 KB
/polyfills.js (localhost)	473 KB	167 KB
/main.js (localhost)	188 KB	31 KB
/styles.js (localhost)	35 KB	11 KB
/runtime.js (localhost)	12 KB	7 KB

Serve images in next-gen formats 0.13 s

Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

Show 3rd-party resources (0)

URL	Size	Potential Savings
 /covid19.png (localhost)	189 KB	167 KB

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

▲ **Serve static assets with an efficient cache policy** — 6 resources found

A long cache lifetime can speed up repeat visits to your page. Learn more.

Show 3rd-party resources (0)

URL	Cache TTL	Size
/vendor.js (localhost)	None	4,958 KB
/polyfills.js (localhost)	None	473 KB
/covid19.png (localhost)	None	189 KB
/main.js (localhost)	None	188 KB

2/7

08/05/2020

URL	Cache TTL	Size
/styles.js (localhost)	None	35 KB
/runtime.js (localhost)	None	12 KB

▲ **Avoid enormous network payloads — Total size was 5,865 KB**

Large network payloads cost users real money and are highly correlated with long load times. Learn more.

Apply route-level code-splitting to minimize the size of your JavaScript bundles. Also, consider precaching assets with the Angular service worker.

☑ Show 3rd-party resources (1)

URL	Size
/vendor.js (localhost)	4,958 KB
/polyfills.js (localhost)	473 KB
/covid19.png (localhost)	189 KB
/main.js (localhost)	188 KB
/styles.js (localhost)	35 KB
/runtime.js (localhost)	12 KB
/countries (api.covid19api.com)	9 KB
/confirmed (localhost)	1 KB
/sockjs-node/info?t=158… (localhost)	0 KB

Reduce JavaScript execution time — 1.4 s

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

☐ Show 3rd-party resources (0)

URL	Total CPU Time	Script Evaluation	Script Parse
/main.js (localhost)	780 ms	763 ms	6 ms
Other	506 ms	20 ms	3 ms
/vendor.js (localhost)	255 ms	5 ms	250 ms
/polyfills.js (localhost)	240 ms	220 ms	12 ms
chrome-extension://fmkadmapgofadopljbjfkapdkoienihi/build/injectGlobalHook.js	86 ms	71 ms	16 ms

Avoid chaining critical requests — 5 chains found

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more.

3/7

08/05/2020

Maximum critical path latency: **140 ms**

Initial Navigation

/confirmed (localhost)

/runtime.js (localhost) - **20 ms, 12.43 KB**

/polyfills.js (localhost) - **20 ms, 472.7 KB**

/styles.js (localhost) - **20 ms, 34.56 KB**

/vendor.js (localhost) - **110 ms, 4,957.91 KB**

/main.js (localhost) - **20 ms, 187.81 KB**

User Timing marks and measures — 30 user timings

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

Name	Type	Start Time	Duration
Zone	Measure	81.82 ms	0.37 ms
Zone:ZoneAwarePromise	Measure	82.32 ms	0.43 ms
Zone:fetch	Measure	82.77 ms	0.11 ms
Zone:toString	Measure	82.92 ms	0.07 ms
Zone:util	Measure	83.09 ms	0.05 ms
Zone:timers	Measure	83.15 ms	0.24 ms
Zone:requestAnimationFrame	Measure	83.41 ms	0.11 ms
Zone:blocking	Measure	83.53 ms	0.16 ms
Zone:EventTarget	Measure	83.71 ms	15.18 ms
Zone:on_property	Measure	98.93 ms	14.71 ms
Zone:customElements	Measure	113.68 ms	0.19 ms
Zone:canvas	Measure	113.89 ms	0.13 ms
Zone:XHR	Measure	114.05 ms	0.18 ms
Zone:geolocation	Measure	114.26 ms	0.21 ms
Zone:PromiseRejectionEvent	Measure	114.51 ms	0.11 ms
Zone	Mark	81.82 ms	
Zone:ZoneAwarePromise	Mark	82.32 ms	
Zone:fetch	Mark	82.78 ms	
Zone:toString	Mark	82.93 ms	
Zone:util	Mark	83.09 ms	
Zone:timers	Mark	83.16 ms	
Zone:requestAnimationFrame	Mark	83.42 ms	
Zone:blocking	Mark	83.54 ms	

4/7

08/05/2020

Name	Type	Start Time	Duration
Zone:EventTarget	Mark	83.72 ms	
Zone:on_property	Mark	98.94 ms	
Zone:customElements	Mark	113.69 ms	
Zone:canvas	Mark	113.9 ms	
Zone:XHR	Mark	114.06 ms	
Zone:geolocation	Mark	114.26 ms	
Zone:PromiseRejectionEvent	Mark	114.52 ms	

Keep request counts low and transfer sizes small — 9 requests • 5,865 KB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	9	5,865 KB
Script	5	5,665 KB
Image	1	189 KB
Other	2	10 KB
Document	1	1 KB
Stylesheet	0	0 KB
Media	0	0 KB
Font	0	0 KB
Third-party	1	9 KB

Passed audits (15)

Eliminate render-blocking resources

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more.

Properly size images

Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

Consider using the `BreakpointObserver` utility in the Component Dev Kit (CDK) to manage image breakpoints. Learn more.

Defer offscreen images

Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

Minify CSS

5/7

08/05/2020

Minifying CSS files can reduce network payload sizes. Learn more.

If you are using Angular CLI, ensure that builds are generated in production mode. Learn more.

Remove unused CSS

Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more.

Efficiently encode images

Optimized images load faster and consume less cellular data. Learn more.

Preconnect to required origins

Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn more.

Server response times are low (TTFB) — Root document took 0 ms

Time To First Byte identifies the time at which your server sends a response. Learn more.

Avoid multiple page redirects

Redirects introduce additional delays before the page can be loaded. Learn more.

Preload key requests

Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more.

Preload routes ahead of time to speed up navigation. Learn more.

Use video formats for animated content

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

Avoids an excessive DOM size — 278 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

Consider virtual scrolling with the Component Dev Kit (CDK) if very large lists are being rendered. Learn more.

Statistic	Element	Value
Total DOM Elements		278
Maximum DOM Depth	`<a _ngcontent-kva-c1="" href="/confirmed">`	8
Maximum Child Elements	`<select _ngcontent-kva-c5="" class="MainDropdown">`	249

Minimizes main-thread work — 1.9 s

6/7