Onderzoeksopdracht Project: Asynchrone functies

```
async function boom() {
  try {
    const fireworks = await getFireworks();
    const trigger = await setUpFireworks(fireworks);
    return trigger();
  } catch(error) {
    return 'Run Away!'
  }
}
```

Klas & Groep: HBO-ICT SE 2B, Projectgroep 04
Vak: Project Amazonsimulatie
Auteurs:

- Damiaen Toussaint (4554752) (damiaen.toussaint@student.nhlstenden.com)
- Mathijs Grafhorst (…) (mathijs.grafhorst@student.nhlstenden.com)

Datum laatste wijziging: 01-11-2019

Inhoudsopgave

1. Inleiding

Dit document betreft de groepsonderzoeksopdracht voor het project Amazonsimulatie. Deze onderzoeksopdracht is onderdeel van het Amazon simulatie project. Dit document is eigendom van team 4 van de klas HBO-ICT SE 2B. Projectgroep 4 bevat Damiaen Toussaint en Mathijs Grafhorst als teamleden.

Tijdens de ontwikkeling van het project liepen we tegen problemen aan met het synchroniseren van de Java backend met de Javascript front-end. Als de front-end applicatie een commando van de backend kreeg, wachtte de front-end niet tot het vorige commando was afgerond. Door dit probleem kregen we rare bugs, zoals het 2 keer plaatsen van een robot of airship waar er eigenlijk maar 1 van moet zijn. Een voorbeeld van één van deze bugs is in de afbeelding hieronder te vinden (Figuur).

Figuur 1 – Bug die er voor zorgt dat er te veel Airships ingeladen worden

Om dit probleem te kunnen oplossen moet er in de code een check komen die afwacht tot het vorige commando afgerond is. Omdat ons team al ervaring heeft met programmeren wisten we dat we iets moesten doen met Promises of Async/Await functies. Maar hoe we deze functionaliteit moeten implementeren in een Javascript applicatie moesten we nog onderzoeken.

2. Opzet van het onderzoek

De onderzoeksmethode die voor dit onderzoek gebruikt gaat worden is de “**Deskresearch**” onderzoeksmethode. In het geval van dit onderzoek weten wij al naar wat voor technieken we moeten zoeken. We zien in dit geval ook niet het nut in van het maken van een mind-map. In de onderstaande opsomming zijn de hoofd- en deel vragen te vinden.

Hoofdvraag van dit onderzoek:

- Hoe implementeren wij een functie die afwacht tot een commando afgehandeld is.

Deelvragen van dit onderzoek:

- Wat voor mogelijkheden zijn er beschikbaar in Javascript.
- Wat zijn de voor en nadelen van deze technieken.
- Wat past het beste in het huidige project.

Doelstelling

De doelstelling van dit onderzoek is het vinden van een techniek die wij kunnen implementeren in ons project. Op basis van de informatie die uit de deelvragen komt komen we uit tot 1 techniek die gaan gebruiken in ons project.

3. Data verzameling en resultaten

Voor het beantwoorden van de deelvragen van dit onderzoek is er met behulp van het internet gekeken naar verschillende bronnen die gaan over Javascript. Er is specifiek gezocht naar bronnen die gaan over de verschillende technieken die synchroon en asynchroon werken, vanuit deze informatie zijn er de volgenden resultaten naar boven gekomen:

Voor Javascript zijn er twee technieken beschikbaar voor het oplossen van het synchronisatieprobleem. Deze technieken zijn Async/Await en Promises, van deze twee technieken is Async/Await de nieuwste. Er zijn verschillende voordelen aan Async/Await in vergelijking met Promises. Zo is de code van een Async functie korter en overzichtelijker dan een Promise. Ook het afhandelen van foutenmeldingen in code gaat een stuk netter en is minder code dan een Promise (Gaafar, 2017).

Hieronder is een voorbeeld te zien van het verschil tussen Javascript code die gebruik maakt van Promises en Async/Await (Figuur-2). Het eerste wat opvalt is dat Async functie aan de rechterkant minder tekst is dan de Promise aan de linkerkant. Een groot voordeel is dus de leesbaarheid van code op de rechterkant (Gaafar, 2017).

```
function logFetch(url) {
  return fetch(url)
    .then(response => response.text())
    .then(text => {
      console.log(text);
    }).catch(err => {
      console.error('fetch failed', err);
    });
}
```

```
async function logFetch(url) {
  try {
    const response = await fetch(url);
    console.log(await response.text());
  }
  catch (err) {
    console.log('fetch failed', err);
  }
}
```

Figuur 2 – Links een functie met een promise, rechts een functie met async

Een ander belangrijk punt is de beschikbaarheid van deze technieken in de browser, het moet natuurlijk wel werken. Als we kijken naar de support van Async/Await implementaties komt er naar boven dat deze techniek op bijna alle browsers werkt, met uitzondering van Internet Explorer (Mozilla, 2019a).

Promises werken ook niet in de Internet Explorer browser, maar er zijn ook verschillende onderdelen van Promises die niet werken in andere browsers. Ook zijn er browsers die andere eisen stellen aan de implementatie van een Promise (Mozilla, 2019a).

4. Rapportage & conclusie

Uit de verkregen informatie zijn er twee technieken naar boven gekomen die toepasbaar zijn in dit project. Dit zijn het implementeren van een Async/Await functionaliteit of Promises gebruiken. Deze twee technieken werken beide op de nieuwste browsers met uitzondering van Internet Explorer. Het laten werken van het project in Internet Explorer is niet van toepassing bij dit project.

Van deze technieken is de Async methode een van de nieuwste en ook de meest uitgebreide van de twee. Verder is de code van een Async methode een stuk korter dan die van een Promise, wat er ook voor zorgt dat de code in het algemeen een stuk overzichtelijker wordt en een stuk netter.

De voorkeur van techniek gaat in ons team uit naar het nieuwste en meest uitgebreide. Dat is in dit geval de Async/Await functionaliteit.

Hoe implementeren wij een functie die afwacht tot een commando afgehandeld is.

Om een functie te kunnen implementeren die ervoor zorgt die afwacht tot een commando afgehandeld is kunnen we kiezen uit Async/Await of Promises. Het mooie aan deze twee technieken is dat je ze ook beide kan implementeren, omdat deze twee technieken samen ook werken. Bij het toevoegen van een van deze functies zal de code afwachten totdat het eerste commando afgerond is en dan pas verder gaan.

Conclusie

Om de bugs op te lossen die code te vaak uitvoert zonder een ander stuk af te wachten hebben we besloten om Promises samen met Async/Await functionaliteit te implementeren. Omdat een Await ook een promise kan afwachten kunnen we beide implementeren.

5. Bronvermelding

Gaafar, M. (2017, 17 maart). *6 Reasons Why JavaScript Async/Await Blows Promises Away (Tutorial)*. Geraadpleegd op 31 oktober 2019, van https://hackernoon.com/6-reasons-why-javascripts-async-await-blows-promises-away-tutorial-c7ec10518dd9

Mozilla. (2019a, 25 juli). *MDN Web Docs - Async function*. Geraadpleegd op 31 oktober 2019, van https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Mozilla. (2019b, 10 augustus). *MDN Web Docs - Promise*. Geraadpleegd op 31 oktober 2019, van https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise