Taakgroep: Javascript AJAX

Taak: Promises in plaats van callbacks

In deze opdracht vervang je het meegeven van een callback door het gebruik van een Promise. De reden hiervoor is dat in de praktijk blijkt dat callbacks genest moeten worden meegegeven, dit leidt tot een callback hell. Het alternatief is om async function met behulp van await op te vangen. Dit kan niet altijd, dan maken we gebruik van then.

Aanpak

Je maakt stapsgewijs kennis met het Promise object toont de opgehaalde gegevens in de console.

- Voer onderstaande code uit in een index.html. Vergeet niet je eigen apikey te gebruiken.

```
const url = 'http://api.openweathermap.org/data/2.5/weather?q=zwolle&apikey=<apikey>
const promise = $.get(url);
console.log(promise);
```

- Voorgaande code roept `$.get` aan, de methode retourneert onmiddelijk een object. Variabele `promise` blijkt een Promise te zijn, terwijl async methode ondertussen nog wordt uitgevoerd.
- De volgende stap is om de data te ontvangen zodra de data is ontvangen. Voer onderstaande code uit en bestudeer de output in de console:

```
    const url = 'http://api.openweathermap.org/data/2.5/weather?q=zwolle&apikey=<apikey>

    $.get(url).then(function(responseData){
        console.log(responseData);
    });
};
```

- Neem onderstaande code over en voer deze uit in de browser.

```
const getWeatherData = function(){
    const url = 'http://api.openweathermap.org/data/2.5/weather?q=zwolle&apikey=<apikey>
    return $.get(url);

};

getWeatherData().then(d => {
    console.log(d);
});
```

- Het is mogelijk om promises te chainen, voer onderstaande code uit in de browser:

```
getWeatherData().then(d => {
        return {
            data: d,
            meta: 'vanuit eerste schakel van de chain'
        }
    }).then(r => {
        console.log(`Bericht ontvangen in tweede schakel: ${r.meta}`);
        throw new Error('Error gegooid');
    }).catch(e => {
        console.log(`Error opgevangen, error bericht: ${e.message}`);
    });
```

Ondersteunende informatie

Promises

Een promise is een object dat gebruikt wordt in geval van async code, zodra deze code is voltooid dan worden de gegevens doorgegeven (resolve).
Op MDN is een uitgebreide beschrijving.

Promise resolve

Een promise geeft een waarde terug door de resolve functie aan te roepen:

```
const p = new Promise((resolve, reject)=>{resolve('klaar')});
```

Om de waarde die resolved is te kunnen gebruiken zijn er doorgaans twee methoden:

```
const r = await functionToResolve();
```

óf

```
functionToResolve.then(r => {

 // do r things

})
```

Promise reject

Een promise kan falen door de reject functie aan te roepen:

```
const p = new Promise((resolve, reject)=>{reject('faal')});
```

Er zijn twee manierne om een reject op te vangen.

Een reject wordt opgevangen in een try catch block:

```
try {
const r = await functionToResolve();
} catch(err){
  //handle err
}
```

óf

```
functionToResolve.then(r => {

 // do r things

}).catch(err => {

//handle err

})
```