



Events JSON – Fetch

Labo 02

Content

LEERDOELEN.....	4
1 EVENTS.....	5
1.1 Event handler attributes in HTML – Bad Bad Bad ❌	5
1.1.1 Inline javascript ❌	5
1.1.2 Oproepen van een function ❌	6
1.2 Events listener ✅	7
1.2.1 Events listener - Good 👍	7
1.2.2 Events listener - Better 👍👍	8
1.2.3 Events listener - Perfect 👍👍	10
2 JSON	12
2.1 Wat is JSON	12
2.2 JSON Object.....	12
2.3 JSON Array.....	12
2.3.1 JSON Array gecombineerd met JSON Array	13
2.4 Javascript en JSON.....	14
2.4.1 JSON (object) via een string.....	14
2.4.2 JSON (object) uit een file	15
2.4.3 JSON (object) uit een file - Laden via een externe bron	15
2.4.4 JSON (object) uit een file - Laden via een interne bron.....	18
2.4.5 JSON via een API endpoint	19
3 FETCH() METHODE.....	25
3.1 Wat is een promise?.....	25
3.1.1 De promise van de fetch-functie	25
3.2 Chained	25
3.2.1 Chained toegepast op fetch-function en json-function	26
3.3 De fetch() methode error proof maken	27
3.3.1 Situatie	27
3.3.2 Oplossing 🙌	27
3.4 De arrow functie (ter info)	28
4 LABO OEFENINGEN.....	30
4.1 Postman interpretatie van JSONObject	30
4.1.1 API: Openlibrary.org	30
4.1.2 API: swapi.co	31
4.1.3 API: opentdb.info	31
4.1.4 API: tvmaze.com	31
4.2 Chrome devTool toepassing.....	32
4.2.1 Opdracht	32
4.2.2 Stappenplan	32

4.3	Afvalcontainers van Antwerpen	33
4.3.1	Opdracht	33
4.3.2	Stappenplan	33
4.3.3	Afwerking	34
4.4	Weather	35
4.4.1	Vooraf.....	35
4.4.2	Opdracht	35
4.4.3	Stappenplan	36
4.4.4	Tips voor deze oefening	36
4.5	Thuisopdracht: Chrome devTool – Deelauto's	37
4.5.1	Opdracht	37
4.5.2	Stappenplan	37
4.6	Thuisopdracht: Chrome devTool – Open Library	38
4.6.1	Opdracht	38
4.6.2	Stappenplan	38
4.6.3	Tip voor deze oefening.....	39
4.7	Thuisopdracht: Uitbreiding weather	39
4.7.1	Opdracht	39
4.8	Thuisopdracht: Bluebike.....	40
4.8.1	Opdracht	40
4.8.2	Stappenplan	40

Leerdoelen

- Gebruik van eventhandlers in Javascript.
- Gebruik van Postman om een response van een API te verkennen.
- Fetch gebruiken om data op te vragen uit een JSON-bestand of API endpoint.
- De verschillende gegevenstypes van een JSON-object herkennen en aanspreken.
- Een JSON object overlopen en de inhoud tonen in Chrome devtool.
- Chrome devtool gebruiken om te debuggen. (Breakpoint en Step Into)
- JSON object overlopen en de inhoud, via de innerHTML van een HTML-element, weergeven.

1 Events

Events zijn *gebeurtenissen* die kunnen optreden op een HTML-element of het HTML-document.

Een event kan iets zijn dat met de browser gebeurt of iets dat je gebruiker uitvoert.

Enkele voorbeelden van events.

- Een HTML-webpagina en alle CSS bestanden, afbeeldingen, scriptbestanden,... zijn geladen. (*load*)
- Enkel de volledige HTML DOM is geladen (zonder afbeeldingen, scripts, CSS,...) (*DOMContentLoaded*)
- Een HTML-element is aangeklikt (*click*)
- Een HTML-element is aangeduid met muis (*mouseover*)

Javascript laat je reageren als zo'n event optreedt.

Er zijn twee manieren om met events te werken in Javascript.

- De "slechte" manier : via het Eventhandler attribute van een HTML element (inline en via functie) in het HTML bestand.
- De "goede" manier : via EventListeners in het Javascript bestand.

1.1 Event handler attributes in HTML – Bad Bad Bad

We kunnen Javascriptcode inline koppelen aan een HTML element via het **event handler attribute** van het HTML element.

Ondanks dat je deze codevoorbeelden online terug vindt, vermijden we het gebruik hiervan in de cursus Full-stack Web Development.

We verkiezen om de programmeercode af te zonderen van het HTML-document.

Voor de volledigheid bekijken we toch twee codevoorbeelden.

1.1.1 Inline javascript

Een voorbeeld waar we:

- Bij het aanklikken de innerHTML van de kop veranderen.
- Wanneer we met de muis over de kop bewegen, verandert de inhoud van het class-attribute van het element naar 'opvallen', zodat de achtergrond rood wordt.
- Wanneer we met de muis de kop verlaten verandert de inhoud van het class-attribute van het element naar '' (de lege string) zodat de achtergrond zijn oorspronkelijke waarde krijgt.
- Via het keyword *this* spreken we het element zelf aan.

inline.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Events demo</title>
    <link rel="stylesheet" type="text/css" href="css/screen.css" />
  </head>
  <body>
    <h1>Bad Bad Bad Boy! Don't try this at home</h1>
    <h2 onclick="this.innerHTML='geklikt'" onmouseover="this.setAttribute('class','u-opvallen')" onmouseout="this.setAttribute('class','')">
      Titel 1a - mouse over - html event handler attributes
    </h2>
  </body>
</html>
```

DON'T TRY THIS AT HOME!

css/screen.css

```
.u-opvallen {
  background-color: #F00
}
```

1.1.2 Oproepen van een functie

Een voorbeeld waar we:

- Bij het aanklikken, de innerHTML van de kop veranderen via een zelfgeschreven functie.
 - Het keyword *this* verwijst binnenin deze javascript functie niet naar het html-element maar naar het *window* object.
 - Willen we het HTML-element aanspreken dat het event aanriep?
Dan moeten we *this* meegeven als parameter van de functie.

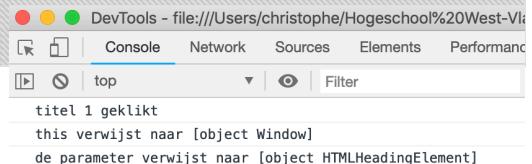
inline_functie.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Events demo</title>
    <script src="script/event_inline.js"></script>
  </head>
  <body>
    <h1>Bad Bad Bad Boy! Don't try this at home</h1>
    <h2 onclick="titel1bGeklikt(this)">
      Titel 1b - click me please... - html event handler attributes + function
    </h2>
  </body>
</html>
```

DON'T TRY THIS AT HOME!

script/event_inline.js

```
const titel1bGeklikt = function(element) {  
    console.log('titel 1 geklikt');  
    console.log('this verwijst naar ' + this);  
    console.log('de parameter verwijst naar ' + element);  
    element.innerHTML = 'je klikt';  
};
```



1.2 Events listener ✓

De beste manier om te werken met events in Javascript is gebruik maken van de `addEventListener` methode. Op deze manier mixen we de programmeercode niet met de HTML code. In de cursus Full-stack Web Development verkiezen we dan ook deze werkwijze.

1.2.1 Events listener - Good 👍

Deze werkwijze heeft één groot nadeel, we moeten het script onderaan onze HTML- pagina laden. Pas op deze plaats zijn we zeker dat de volledige DOM geladen is.

In hoofdstuk 1.2.3 *Events listener - Perfect* 🎉 zien we hiervoor een oplossing.

```
htmlElement.addEventListener('naamEvent', function(par_event) {  
});
```

eventlistener_good.html – css/screen.css

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <title>Events demo</title>  
    <link rel="stylesheet" type="text/css" href="css/screen.css" />  
  </head>  
  <body>  
    <h1>That's better</h1>  
    <h2 class="js-title2a">  
      Titel 2a - mouse over/click - addEventListener - anonymos function  
    </h2>  
  
    <h1>Output</h1>  
    <div class="js-output"></div>  
  
    <script src="script/event_good.js"></script>  
  </body>  
</html>
```

script/event_good.js

- We declareren twee constanten die de HTML-elementen voorstellen (de kop en de plaats waar de output getoond wordt).
- Via de `addEventListener`-methode "binden" we onze code aan het het `click`, `mouseover`- en `mouseout`-event.
 - De eerste parameter is de naam van het event waaraan we onze code willen koppelen.
 - De tweede functie is een anonieme functie waarbinnen we onze code noteren.

- We zien dat we optioneel 1 parameter (in ons voorbeeld *e*) kunnen opvragen. Deze parameter stelt het event voor, met bijhorende eigenschappen.
- Om naar het html-element te verwijzen, die het event opriep, kunnen we het keyword *this* gebruiken in onze code.

```
const htmlTitle2a = document.querySelector('.js-title2a');
const htmlOutput = document.querySelector('.js-output');

htmlTitle2a.addEventListener('click', function(e) {
  console.log('Titel 2a geklikt');
  console.log('this verwijst naar ' + this);
  console.log('de parameter verwijst naar ' + e);
  console.log(e)

  htmlOutput.innerHTML =
    'we riepen het click event aan via eventlistener, de parameter e moet niet worden
    opgegeven, zoals je kan zien bij mouseover/mouseout';
});

htmlTitle2a.addEventListener('mouseover', function() {
  console.log('Titel 2a mouseover');
  console.log('this verwijst naar ' + this);
  this.setAttribute('class', 'u-opvallen');

  htmlOutput.innerHTML =
    'we riepen het mouseover event aan via eventlistener, de parameter e moet niet
    worden opgegeven';
});

htmlTitle2a.addEventListener('mouseout', function() {
  console.log('Titel 2a mouseout');
  console.log('this verwijst naar ' + this);
  this.setAttribute('class', '');

  htmlOutput.innerHTML =
    'we riepen het mouseout event aan via eventlistener, de parameter e moet niet
    worden opgegeven';
});
```

Titel 2a geklikt
this verwijst naar [object HTMLElement]
de parameter verwijst naar [object MouseEvent]
▼ MouseEvent {isTrusted: true, screenX: -1393, screenY: 283, clientX: 236, clientY: 100, ...}

altKey: false
bubbles: true
button: 0
buttons: 0
cancelBubble: false
cancelable: true
clientX: 236

Titel 2a mouseout
this verwijst naar [object HTMLElement]

1.2.2 Events listener - Better👍👍

We merken op dat de inhoud van onze anonymous function snel overladen wordt en de code niet eenvoudig te hergebruiken is. Een betere manier is om een **eigen functie** op te roepen als tweede parameter van de *addEventListener* methode. Belangrijk is dat we enkel de naam gebruiken van onze eigen functie.

Je plaatst geen extra haakjes, of geeft op deze plaats geen extra parameters op!

Deze werkwijze blijft één groot nadeel houden, we moeten het script onderaan onze html pagina laden. Pas daar zijn we zeker dat de volledige DOM geladen is. In hoofdstuk 1.2.3 Events listener - Perfect👌 zie we hiervoor een oplossing.

```
htmlElement.addEventListener('naamEvent',naamFunctie);
```

eventlistener_better.html – css/screen.css

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Events demo</title>
    <link rel="stylesheet" type="text/css" href="css/screen.css" />
  </head>
  <body>
    <h1>That's better</h1>
    <h2 class="js-title2b">
      Titel 2b - mouse over/click - addEventListener - eigen functie
    </h2>

    <h1>Output</h1>
    <div class="js-output"></div>

    <script src="script/event_good.js"></script>
  </body>
</html>
```

script/event_better.js

- We declareren twee constanten die de html-elementen voorstellen (de knop en de plaats van output).
- Via de *addEventListener* methode "binden" we onze code aan het *click-*, *mouseover-* en *mouseout-*event.
 - De eerste parameter is de naam van het event waaraan we onze code willen koppelen.
 - De tweede functie is de naam van onze eigen functies die we willen koppelen.
- Belangrijk is dat we onze eigen functies BOVEN de *addEventListener*s noteren.
 - We kunnen bij de eigen functie optioneel 1 parameter noteren die het event voorstelt.
 - Via *e.type* kan bijvoorbeeld het eventtype worden opgevraagd.
- Om naar het html-element te verwijzen die het event opriep, kunnen we het *this* keyword gebruiken.

Tip:

- Via de *removeEventListener* methode kunnen we een *EventListener* verwijderen.
 - In onderstaand voorbeeld wordt de *EventListener mouseover-veranderAchtergrondkleur()* verwijderd.
- We zien dat we via *addEventListener* meerdere acties kunnen koppelen aan één event.
 - In onderstaand voorbeeld wordt zowel *toonGegevens()* als *veranderAchtergrondkleur()* uitgevoerd bij het *mouseover*-event.

```

const htmlTitle2b = document.querySelector('.js-title2b');
const htmlOutput = document.querySelector('.js-output');

const toonLogGegevens = function(e) {
    console.log('Actie op Titel 2b ');
    console.log('this verwijst naar ' + this);
    console.log('de parameter verwijst naar ' + e);
    console.log(
        'om "caller" op te roepen gebruik e.target' +
        e.target +
        ' dit is gelijk aan this ' +
        this
    );
    htmlOutput.innerHTML +=`${this.id} is ${e.type} <br/>`;
};

const veranderAchtergrondkleur = function() {
    this.setAttribute('class', 'u-opvallen');
    htmlOutput.innerHTML +=`${this.id} is veranderd van kleur<br/>`;
};

const resetAchtergrondkleur = function() {
    this.setAttribute('class', '');
    htmlOutput.innerHTML +=`${this.id} is originele kleur<br/>`;
    this.removeEventListener('mouseover', veranderAchtergrondkleur);
    htmlOutput.innerHTML +=`${this.id} mouse over event is verwijderd<br/>`;
};

/* belangrijk noteer de functies BOVEN de eventListeners */
htmlTitle2b.addEventListener('click', toonLogGegevens);
htmlTitle2b.addEventListener('mouseover', toonLogGegevens);
htmlTitle2b.addEventListener('mouseover', veranderAchtergrondkleur);
htmlTitle2b.addEventListener('mouseout', toonLogGegevens);
htmlTitle2b.addEventListener('mouseout', resetAchtergrondkleur);

```

1.2.3 Events listener - Perfect👌

In de laatste twee voorbeelden hadden we steeds het probleem dat de javascript verwijzingen onderaan de html code werden geplaatst. Dit was omdat we pas op deze plaats zeker waren dat de volledige DOM was geladen, en we in de javascript code pas op dat moment veilig konden verwijzen naar het html-element.

Het zou beter zijn om de vorige code te combineren met het *Load* event van onze pagina. Er zijn twee interessante events waar we onze code aan kunnen koppelen.

Het *load*-event en *DOMContentLoaded*-event.

Het *DOMContentLoaded*-event is de beste keuze omdat deze wordt uitgevoerd nadat de html-DOM is geladen.

Het *Load*-event werkt trager omdat dit pas wordt uitgevoerd als én de HTML-dom én alle externe files (jpg's, css, js,...) geladen zijn.

eventlistener_perfect.html – css/screen.css

Merk op dat de verwijzing naar de Javascript file nu naar de header is verhuisd.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Events demo</title>
    <link rel="stylesheet" type="text/css" href="css/screen.css" />
    <script src="script/event_perfect.js"></script>
  </head>
  <body>
    <h1>This is perfect</h1>

    <h2 class="js-title2c">
      Titel 2c - mouse over/click - addEventListener - eigen functie - na DOMLoaded event
    </h2>

    <h1>Output</h1>
    <div class="js-output"></div>
  </body>
</html>
```

script/event_perfect.js

In onderstaand voorbeeld zien we een combinatie van anonymous functie en eigen functies. Omdat we met het *DOMContentLoaded* event werken, kunnen we in eventlistener_perfect.html de verwijzing naar het javascript document in de header plaatsen.

```
const toonLogGegevens = function(e) {
  console.log('Actie op Titel 2b ');
  console.log('this verwijst naar ' + this);
  console.log('de parameter verwijst naar ' + e);
  console.log('om "caller" op te roepen gebruik e.target' + e.target + ' dit is gelijk aan this ' + this);
  const htmlOutput = document.querySelector('.js-output');
  htmlOutput.innerHTML += `${this.id} is ${e.type} <br/>`;
};

const veranderAchtergrondkleur = function() {
  this.setAttribute('class', 'u-opvallen');
  const htmlOutput = document.querySelector('#output');
  htmlOutput.innerHTML += `${this.id} is veranderd van kleur<br/>`;
};

const resetAchtergrondkleur = function() {
  this.setAttribute('class', '');
  const htmlOutput = document.querySelector('#output');
  htmlOutput.innerHTML += `${this.id} is originele kleur<br/>`;
  this.removeEventListener('mouseover', veranderAchtergrondkleur);
  htmlOutput.innerHTML += `${this.id} mouse over event is verwijderd<br/>`;
};

const koppelEventListeners = function() {
  const htmlTitle2c = document.querySelector('.js-title2c');

  htmlTitle2c.addEventListener('click', toonLogGegevens);
  htmlTitle2c.addEventListener('mouseover', toonLogGegevens);
  htmlTitle2c.addEventListener('mouseover', veranderAchtergrondkleur);
  htmlTitle2c.addEventListener('mouseout', toonLogGegevens);
  htmlTitle2c.addEventListener('mouseout', resetAchtergrondkleur);
};

document.addEventListener('DOMContentLoaded', function() {
  console.info('DOM geladen');
  koppelEventListeners();
});
```

2 Json

2.1 Wat is JSON

JSON of JavaScript Object Notation, is een gestandaardiseerd gegevensformaat. JSON maakt gebruik van - voor de mens - leesbare tekst in de vorm van data-objecten die bestaan uit een of meer **attributen** met bijbehorende **waarden**.

De JSON files worden opgeslagen met de extensie *.json en zijn eigenlijk tekstbestanden.

Een voorbeeld:

inhoud.json

```
{  
    "lesinhoud": "Javascript",  
    "module": "Full-stack Web Development",  
    "dag": "Maandag",  
    "lokaal": "Aula Major",  
    "aantaluur": 4,  
    "eersteJaar": true  
}
```

JSON data wordt genoteerd in name/value paren.

In JSON zijn volgende datatypes toegestaan:

- string
- nummer
- boolean
- null
- array
- (json)object

Wat kan niet in een JSON-file

- functie
- datum
- undefined

2.2 JSON Object

De value kan zelf opnieuw een object zijn. Een (JSON)object wordt opnieuw genoteerd binnen accolades {}.

inhoud.json

```
{  
    "lesinhoud": "Javascript",  
    "module": "Full-stack Web Development",  
    "dag": "Maandag",  
    "lokaal": "Aula Major",  
    "aantaluur": 4,  
    "eersteJaar": true,  
    "moduleVerantwoordelijke": {"naam": "Dieter", "familienaam": "Roobrouck"}  
}
```

2.3 JSON Array

De value kan ook een Array zijn. Een Array kan je vergelijken met een list die we kennen uit Python. Een list wordt genoteerd tussen brackets [].

inhoud.json

```
{  
    "lesinhoud": "Javascript",  
    "module": "Full-stack Web Development",  
    "dag": "Maandag",  
    "lokaal": "Aula Major",  
    "aantalUur": 4,  
    "eersteJaar": true,  
    "moduleVerantwoordelijke": {"naam": "Dieter", "familienaam": "Roobrouck"},  
    "groepen": ["1MCT1", "1MCT2", "1MCT3", "1MCT4", "1MCT5", "1MCT6"]  
}
```

Binnen een Array kunnen we alle datatypes gebruiken die zijn toegestaan door JSON. In bovenstaand voorbeeld vullen we de Array op met strings.

2.3.1 JSON Array gecombineerd met JSON Array

Wanneer we meerdere objecten binnen eenzelfde attribute willen opslaan, volstaat het om de objecten in een Array onder te brengen.

In onderstaand voorbeeld plaatsen we, als array, twee (json)objecten - die een lector voorstellen - in het attribute lectoren.

inhoud.json

```
{  
    "lesinhoud": "Javascript",  
    "module": "Full-stack Web Development",  
    "dag": "Maandag",  
    "lokaal": "Aula Major",  
    "aantalUur": 4,  
    "eersteJaar": true,  
    "moduleVerantwoordelijke": {"naam": "Dieter", "familienaam": "Roobrouck"},  
    "groepen": ["1MCT1", "1MCT2", "1MCT3", "1MCT4", "1MCT5", "1MCT6"],  
    "lectoren": [{"naam": "Frederik", "familienaam": "Waeyaert"}, {"naam": "Christophe", "familienaam": "Laprudence"}]  
}
```

Een extra voorbeeld

In onderstaand voorbeeld plaatsen we verschillende (json)objecten die een module voorstellen, in een (json)Array MCT.

modules.json:

```
{  
  "MCT": [  
    {  
      "lesinhoud": "Javascript",  
      "module": "Full-stack Web Development",  
      "dag": "Maandag",  
      "lokaal": "Aula Major"  
    },  
    {  
      "lesinhoud": "HTML",  
      "module": "User Interface Design",  
      "dag": "Maandag",  
      "lokaal": "Aula Minor"  
    }  
,  
  "Devine": [  
    {  
      "lesinhoud": "Creatief skills",  
      "module": "C01",  
      "dag": "Woensdag",  
      "lokaal": "B0.008"  
    }  
]  
}
```

2.4 Javascript en JSON

Wanneer we in Javascript een object aanmaken kunnen we dit doen door

- Een variabele te declareren als object en hierin een string in te lezen (via `JSON.parse()`).
- Een (lokaal of extern) JSON bestand in te lezen via `Fetch()`.
- Een (extern) endpoint van een REST API aan te spreken via `Fetch()`.

2.4.1 JSON (object) via een string

Het JSON object is een string dat wordt omgezet naar een Javascript Object met behulp van `JSON.parse()`. Deze string is “hard coded”, de inhoud staat vast genoteerd in de javascript file.

agenda.html - script/agenda.js

```
let lesrooster = '{"lesinhoud": "Javascript", "module": "Full-stack Web Development", "dag": "Maandag", "lokaal": "Aula Major"}';  
  
let lessen = JSON.parse(lesrooster);
```

Na de creatie van het object kan je via de **key** van de **attributen** de **values** opvragen. Op deze manier kunnen we de inhoud (de value) uitlezen. We kunnen de value van een attribute op 2 manieren opvragen.

```
let agendapunt = lessen.lesinhoud;  
/* of */  
let agendapunt = lessen['lesinhoud']
```

```

let lesrooster = '{"lesinhoud": "Javascript", "module": "Full-stack Web Development", "dag": "Maandag", "lokaal": "Aula Major"}';

let lessen = JSON.parse(lesrooster);

console.log(`De lessen ${lessen.lesinhoud} gaan door in lokaal ${lessen.lokaal} op ${lessen.dag}`);
console.log(`De lessen ${lessen['module']} gaan door in lokaal ${lessen['lokaal']} op ${lessen['dag']}`);

```

2.4.2 JSON (object) uit een file

Studietip: Leer dit onderdeel samen met hoofdstuk 3 *Fetch() methode*.

In veel gevallen is het niet handig om de JSON data “hard coded” in de javascript file te bewaren. Meestal laden we de data uit een JSONfile (extern of intern) of gebruik je de endpoint van REST API.

We bekijken eerst een voorbeeld waar je data laadt uit een JSONfile. Zo'n bestand is niets meer of minder dan een tekstbestand met 1 stringwaarde in. Deze stringwaarde stelt een JSON object voor.
Als extensie voor dit type bestand gebruiken we meestal *.json.

Het bestand kan zowel op een externe webserver staan (zie 2.4.3) of lokaal op de eigen webserver (zie 2.4.4).

De file inladen doen we via de *fetch()*-methode van javascript. (zie ook hoofdstuk 3 *Fetch() methode*)

2.4.3 JSON (object) uit een file - Laden via een externe bron

 **howest.html - script/howest.js – <http://edu.laprudence.be/fswd/json/howest.json>**

- Header Access-Control-Allow-Origin

Als we een bestand laden van een externe server, moet de externe server dit expliciet toestaan. Een externe server kan aanvragen die van een ander domein afkomstig zijn wijgeren.

In de response header (kan) geeft een server aan hoe hij omgaat met externe aanvragen.

Dit doet de server door in de response de header **Access-Control-Allow-Origin →*** mee te sturen. Onze browser zal controleren of deze header aanwezig is in de response van de server.

Ontbreekt deze header, of is de value van de header ongeldig?

Dan zal de browser ALTIJD weigeren de data te verwerken via de *fetch()* methode!

We kunnen hier als developer niets aan veranderen. Dit is de Cross-Origin Resource Sharing (CORS) beveiliging van de browser.

Lees meer over CORS op <https://medium.com/@abderrahman.hamila/cors-is-not-your-nightmare-but-6cbc749400cf>

De asterisk in **Access-Control-Allow-Origin →*** betekent dat de aanvrager om het even welk domein mag zijn. De externe server kan dit echter ook beperken. Door op te geven welke andere domeinen zijn data mag opvragen via *Fetch()*.

De header van een response kunnen we snel controleren door een GET request naar de server te sturen in Postman.

Via headers tab controleren we of **Access-Control-Allow-Origin** aanwezig is.

GET http://edu.laprudence.be/fswd/json/howest.json

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (11) Test Results Status: 200 OK Time: 103ms Size: 980 B Save Response ▾

KEY	VALUE
Date ⓘ	Thu, 06 Feb 2020 14:39:22 GMT
Server ⓘ	Apache
Last-Modified ⓘ	Thu, 07 Feb 2019 16:23:50 GMT
ETag ⓘ	"25a-58150447ecfff"
Accept-Ranges ⓘ	bytes
Content-Length ⓘ	602
Access-Control-Allow-Origin ⓘ	*
Access-Control-Allow-Headers ⓘ	origin, x-requested-with, content-type
Keep-Alive ⓘ	timeout=5, max=100
Connection ⓘ	Keep-Alive
Content-Type ⓘ	application/json

- Verwerken van de data

```

const verwerkenHowestData = function(jsonHowestData) {
    console.log('*****');
    const inhoud = jsonHowestData.lesinhoud;
    const lokaal = jsonHowestData.lokaal;
    //variabele verantwoordelijke maken en opvullen
    const verantwoordelijke =
        jsonHowestData.moduleVerantwoordelijke.naam + ' ' +
        jsonHowestData.moduleVerantwoordelijke.familienaam;
    //variabele groepen maken en met lus opvullen
    let groepen = '';
    for (const g of jsonHowestData.groepen) {
        groepen += g + ' ';
    }
    //variabele aantal lectoren maken en tellen via functie length
    const aantalExtraLectoren = jsonHowestData.lectoren.length;
    //variabele extralectoren maken en opvullen via lus.
    let extraLect = '';
    for (const lect of jsonHowestData.lectoren) {
        extraLect += lect.familienaam + ' ';
    }
    //uitvoer van inhoud, lokaal, verantwoordelijk en groepen
    console.log(
        `De inhoud vandaag is ${inhoud} in ${lokaal} gegeven door ${verantwoordelijke} voor
        groepen ${groepen}.`);
    );
    //uitvoer van extralectoren en aantal ervan.
    console.log(
        `Labo wordt gegeven door ${aantalExtraLectoren} extra lectoren (${extraLect})`);
};

const laadHowestData = function() {
    //ophalen externe JSON file
    fetch('http://edu.laprudence.be/fswd/json/howest.json')
        .then(function(response)
    {
        //antwoord van de server nakijken op het verzoek
        if (!response.ok) {
            //antwoord is niet ok. error wordt geworpen
            throw Error(`Probleem bij de fetch(). Status Code: ${response.status}`);
        } else {
            //antwoord is ok
            console.info('Er is een response teruggekomen van de server');
            return response.json();
        }
    })
    .then(function(jsonObject) {
        //functie uitgevoerd en json maken
        console.info('json object is aangemaakt');
        //functie verwerkenhowestdata uitvoeren
        verwerkenHowestData(jsonObject);
    })
    //als uitvoeren op een fout loopt
    .catch(function(error) {
        console.error(`fout bij verwerken json ${error}`);
    });
};

//deze regels worden eerst uitgevoerd, alles start vanaf hier.
document.addEventListener('DOMContentLoaded', function() {
    console.info('DOM geladen');
    laadHowestData();
});

```

Query Params	
KEY	VALUE
Key	Value
Body	
Pretty	Raw
JSON	BETA
1	{
2	"lesinhoud": "Javascript",
3	"module": "Full-stack Web Development",
4	"dag": "Maandag",
5	"lokaal": "Aula Major",
6	"aantalUur": 4,
7	"eersteJaar": true,
8	"moduleVerantwoordelijke": {
9	"naam": "Dieter",
10	"familienaam": "Roobrouck"
11	},
12	"groepen": [
13	"1MCT1",
14	"1MCT2",
15	"1MCT3",
16	"1MCT4",
17	"1MCT5",
18	"1MCT6"
19],
20	"lectoren": [
21	{
22	"naam": "Frederic",
23	"familienaam": "Waeyaert"
24	},
25	{
26	"naam": "Christophe",
27	"familienaam": "Laprudence"
28	}
29]
30]

Stappenplan (*lees ook hoofdstuk 3: Fetch() methode*):

- Inladen van de JSON file nadat de *DOMContent* is geladen.
- We hoeven enkel de parameter aan te passen van de *fetch()* methode, met de URL (vergeet het http(s) protocol) van de bron die we willen bevragen.
- Is de *fetch()* en *json()*-methode gelukt? Dan roepen we het deelprobleem *verwerkenHowestData* aan. Het JSON object geven we mee als parameter aan dit deelprobleem.
- Via de for of-structuur overlopen we de Arrays met groepen en lectoren.
- Tip: gebruik Postman om snel een inzicht te krijgen in het JSON object. Op deze manier herkennen we de Arrays en objecten.

```
DOM geladen
Er is een response teruggekomen van de server
json object is aangemaakt
*****
De inhoud vandaag is Javascript in Aula Major gegeven door Dieter Roobrouck voor groepen 1MCT1 1MCT2 1MCT3 1MCT4 1MCT5 1MCT6 .
Labo wordt gegeven door 2 extra lectoren (Waeyaert Laprudence )
```

2.4.4 JSON (object) uit een file - Laden via een interne bron

Belangrijk: door de CORS beveiliging van je browser werkt de *fetch()* methode enkel via een http-request of https-request.

Onderstaand voorbeeld werkt dus niet als we een html bestand openen in VS Code via "Open in default Browser", of door te dubbelklikken op de file in de verkenner.

Aan de URL in de browser kunnen we zien dat het in die situatie wordt geopend via het `file://` protocol. Alle scripts en bestanden die we laden binnen de pagina zullen dus hetzelfde protocol (`file://`) gebruiken. Dus ook een request naar het `*.json` bestand.

Willen we onderstaand voorbeeld testen dan moet dit gebeuren op een (lokale) webserver zoals MAMP/WAMP/XAMPP/Live Server plug-in. Op deze manier wordt de html file, alle scripts en bestanden geladen door een http-request te versturen naar de (lokale) webserver die draait op het adres localhost .



howest.html - script/howest.js - data/howest.json

```
intern
  data
    howest.json
  script
    howest.js
  howest.html
```

```
const verwerkenHowestData = function(jsonHowestData) {
  console.log('*****');
  const inhoud = jsonHowestData.lesinhoud;
  const lokaal = jsonHowestData.lokaal;
  console.log(`De inhoud vandaag is ${inhoud} in ${lokaal}`);
};

const laadHowestData = function() {
  //ophalen interne JSON file
  fetch('./data/howest.json')
    .then(function(response) {
      //antwoord van de server nakijken op het verzoek
      if (!response.ok) {
        //antwoord is niet ok. error wordt geworpen
        throw Error(`Probleem bij de fetch(). Status Code: ${response.status}`);
      } else {
        //antwoord is ok
        console.info(`Er is een response teruggekomen van de server`);
        return response.json();
      }
    })
    .then(function(jsonObject) {
      //functie uitgevoerd en json maken
      console.info('json object is aangemaakt');
      //functie verwerkenhowestdata uitvoeren
      verwerkenHowestData(jsonObject);
    })
    //als uitvoeren op een fout loopt
    .catch(function(error) {
      console.error(`fout bij verwerken json ${error}`);
    });
};

//deze regels worden eerst uitgevoerd, alles start vanaf hier.
document.addEventListener('DOMContentLoaded', function() {
  console.info('DOM geladen');
  laadHowestData();
});
```

We zien dat de code dezelfde opbouw heeft als het vorige voorbeeld. Het enige verschil is dat de parameter van `fetch()` nu relatief verwijst naar de json file op onze eigen webserver. Het protocol moeten we niet opgeven bij een relatieve verwijzing.

2.4.5 JSON via een API endpoint

brouwerij.html – script/brouwerij.js – https://api.openbrewerydb.org/breweries?by_state=california

Veel servers stellen JSON data beschikbaar via een API endpoint. Een endpoint verwijst niet naar een JSON file, maar naar een route op een webserver. Deze route genereert een response met JSON object in de body.

Meestal is zo'n API zodanig geprogrammeerd dat we via extra parameters aan dit endpoint andere data kunnen opvragen.

Vaak moet wel een account worden aangemaakt bij een aanbieder van een bepaalde API (en soms betalen) per request naar deze API. De credentials moeten dan worden meegestuurd met elke request.

Gelukkig bestaan er ook gratis 'Open Data' API's. Deze kan je gebruiken zonder credentials.

Lees altijd de documentatie van de API die je wil aanspreken én controleer via Postman of de **Access-Control-Allow-Origin** header wordt meegestuurd.

Als voorbeeld nemen we de [OpenBreweryDB](#), deze API genereert een lijst van brouwerijen. Als we de documentatie aandachtig lezen zien we hoe we kunnen filteren op Amerikaanse staat.

The screenshot shows the homepage of the Open Brewery DB website. The main content area includes sections for 'About', 'Why', 'Projects', and 'Credits'. The 'About' section provides an overview of the API, stating it's a free API for public information on breweries, cideries, brewpubs, and bottleshops, currently focused on the United States. The 'Why' section expresses the creator's passion for craft breweries and the goal of creating an up-to-date, curated, and publicly available database. The 'Projects' section lists a Python API wrapper maintained by @jrbourbeau. The 'Credits' section acknowledges Chris J Mears and Wandering Leaf Studios LLC, the initial dataset from the Brewers Association, and the logo compiled via Symbolicons.

The screenshot shows the documentation for the 'List Breweries' endpoint. It explains that this endpoint returns a list of breweries. An example output is provided in JSON format, showing a list of brewery objects. One object in the list is highlighted, showing details such as ID, name, type, street address, city, state, and zip code.

```
[  
  ...  
  {  
    id: 399,  
    name: "Almanac Beer Company",  
    brewery_type: "micro",  
    street: "651B W Tower Ave",  
    city: "Alameda",  
    state: "California",  
    zip: "94501"  
  }  
]
```

by_city

Filter breweries by city.

Note: For the parameters, you can use underscores or [url encoding](#) for spaces.

Examples

```
https://api.openbrewerydb.org/breweries?by\_city=san\_diego https://api.openbrewerydb.org/breweries?by\_city=san%20diego
```

by_name

Filter breweries by name.

Note: For the parameters, you can use underscores or [url encoding](#) for spaces.

Examples

```
https://api.openbrewerydb.org/breweries?by\_name=cooper https://api.openbrewerydb.org/breweries?by\_name=modern%20times
```

by_state

Filter breweries by state.

Note: Full state name is required; no abbreviations. For the parameters, you can use underscores or [url encoding](#) for spaces.

Examples

```
https://api.openbrewerydb.org/breweries?by\_state=ohio https://api.openbrewerydb.org/breweries?by\_name=new\_york https://api.openbrewerydb.org/breweries?by\_name=new%20mexico
```

by_postal

Filter breweries by postal code

May be filtered by basic (5 digit) postal code or more precisely filtered by postal+4 (9 digit) code.

Note If filtering by postal+4 the search must include either a hyphen or an underscore.

We beslissen dat we alle brouwerijen van California willen opvragen. Het endpoint zal er dus als volgt uitzien. https://api.openbrewerydb.org/breweries?by_state=california

• Stap 1: Postman – Verken het JSON object

Dit endpoint stuurt een Array terug met (JSON)objecten in. Dit herkennen we doordat de response start met [] en binnen de Array { } staan.

The screenshot shows the Postman interface with a GET request to https://api.openbrewerydb.org/breweries?by_state=california. The response body is displayed in JSON format, showing an array of two brewery objects. The first object is for "7 Sisters Brewing Co" located in San Luis Obispo, California, with coordinates approximately -120.6706375, 35.2467277959184. The second object has an ID of 286.

```
[{"id": 281, "name": "7 Sisters Brewing Co", "brewery_type": "brewpub", "street": "181 Tank Farm Rd Ste 110", "city": "San Luis Obispo", "state": "California", "postal_code": "93401-7082", "country": "United States", "longitude": "-120.670637530612", "latitude": "35.2467277959184", "phone": "8058687133", "website_url": "http://www.7sistersbrewing.com", "updated_at": "2018-08-23T23:23:55.334Z", "tag_list": []}, {"id": 286, "name": "The Beer Project", "brewery_type": "brewpub", "street": "1000 Main St", "city": "Santa Barbara", "state": "California", "postal_code": "93101-2201", "country": "United States", "longitude": "-120.670637530612", "latitude": "35.2467277959184", "phone": "8058687133", "website_url": "http://www.thebeerproject.com", "updated_at": "2018-08-23T23:23:55.334Z", "tag_list": []}]
```

Op het eerste zicht lijkt het endpoint van OpenBreweries niet de juiste header mee te sturen. *Access-Control-Allow-Origin* verschijnt niet tussen de headers in Postman.

Als developer gaan er nu enkele alarmbelletjes af. We kunnen nu enkel nog testen door onze code te schrijven via de *fetch* methode.

Soms aanvaardt een webserver wel een extern request, maar stuurt de webserver de correcte header niet.

Bij het ontbreken van de access-control-allow-origin header, zal de browser ervan uitgaan dat de aanvraag mag doorgaan.

Maar meestal weten we hier al of de request al dan niet zal lukken.

Headers (16)		Status: 200 OK Time: 1103ms Size: 7.72 KB Save Response ▾
KEY	VALUE	
Date ⓘ	Thu, 06 Feb 2020 15:54:07 GMT	
Content-Type ⓘ	application/json; charset=utf-8	
Transfer-Encoding ⓘ	chunked	
Connection ⓘ	keep-alive	
Cache-Control ⓘ	max-age=6400, public	
Etag ⓘ	"fb395912e14613c1a945a556"	
X-Request-Id ⓘ	8502-fc10-4f2a-9944-71240c81f373	
X-Runtime ⓘ	0.129394	
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains	
Vary ⓘ	Origin	
Via ⓘ	1.1 vegur	
CF-Cache-Status ⓘ	DYNAMIC	
Expect-CT ⓘ	max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi...	
Server ⓘ	cloudflare	
CF-RAY ⓘ	560e4d024f13ee1b-CDG	
Content-Encoding ⓘ	br	

??Access-Control-Allow-Origin ??

- **Stap 2 – Schrijf basis structuur voor Fetch methode**

Roep via de *fetch()*-functie je endpoint van de API aan. Omdat we niet zeker waren dat de juiste header werd meegestuurd zijn er twee mogelijkheden.

- De webserver weigert requests van externe domeinen
 - De status van de response zal **niet “ok”** zijn.
Hier kan je niets aan veranderen. De API is niet bereikbaar via *Fetch()*.
- De webserver aanvaardt toch requests van externe domeinen
 - De response.status zal **“ok”** zijn en je kan doorgaan met coderen.

script/brouwerij.js

```
const verwerkenBrouwerij = function(jsonBrouwerij) {
    console.log('start verwerken brouwerij data');
};

const laadAPIData = function() {
    //ophalen API data
    fetch('https://api.openbrewerydb.org/breweries?by_state=california')
        .then(function(response) {
            //antwoord van de server nakijken op het verzoek
            if (!response.ok) {
                //antwoord is niet ok. error wordt geworpen
                throw Error(`Probleem bij de fetch(). Status Code: ${response.status}`);
            } else {
                //antwoord is ok
                console.info('Er is een response teruggekomen van de server');
                return response.json();
            }
        })
        .then(function(jsonObject) {
            //functie uitgevoerd en json maken
            console.info('json object is aangemaakt');
            //functie verwerkenhowestdata uitvoeren
            verwerkenBrouwerij(jsonObject); /* zie stap 3 */
        })
        //als uitvoeren op een fout loopt
        .catch(function(error) {
            console.error(`fout bij verwerken json ${error}`);
        });
};

//deze regels worden eerst uitgevoerd, alles start vanaf hier.
document.addEventListener('DOMContentLoaded', function() {
    console.info('DOM geladen');
    laadAPIData();
});
```

We hebben geluk. We krijgen een response.ok terug van de openbrewery webserver.

- **Stap 3 – Values uit JSON object tonen in het HTML bestand**

We werken het deelprobleem *verwerkenBrouwerij()* verder af.

Ditmaal tonen we de uitvoer niet via *console.log()* maar vervangen we de *innerHTML* (HTML inhoud) van het element met als class "js-placeholder".

Hiervoor declareren we een variabele *htmlPlaceholder*, die het HTML element met class "js-placeholder" vertegenwoordigt in de javascriptcode. Hiervoor gebruiken we de *querySelector()* functie.

Vervolgens bouwen we een string op die de Array van brouwerijen overloopt. Uiteindelijk willen we volgende string bekomen.

```
<ul>
    <li>7 Sisters Brewing Co - (San Luis Obispo)</li>
    <li>Naam - (Stad)</li>
    <li>...</li>
</ul>
```

herhaling

Tenlaatste vervangen we de *innerHTML* van het HTML element met class js-placeholder.

script/brouwerij.js

```
...
const verwerkenBrouwerij = function(jsonBrouwerij) {
    console.log('start verwerken brouwerij data');
    /* de variabele htmlPlaceholder stelt het DOM element met class js-placeholder voor */
}
const htmlPlaceholder = document.querySelector('.js-placeholder');
/* Pas de HTML code aan die binnen deze tag staat */
let inhoud = '';
inhoud = '<ul>';
for (const brouwerij of jsonBrouwerij) {
    inhoud += `<li>${brouwerij.name} - (${brouwerij.city})</li>`;
}
inhoud += '</ul>';
htmlPlaceholder.innerHTML = inhoud;
};
```

herhaling

brouwerij.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Title of the
document</title>
    <script
src="script/brouwerij.js"></script>
  </head>

  <body>
    <h1>Lijst van brouwerijen</h1>
    <div class="js-placeholder">
      loading...
    </div>
  </body>
</html>
```

Lijst van brouwerijen

- 7 Sisters Brewing Co - (San Luis Obispo)
- Abnormal Beer Company - (San Diego)
- Altamont Beer Works - (Livermore)
- Auburn Alehouse - (Auburn)
- Back Street Brewery & Tasting Room - (Anaheim)
- Bear Republic Brewing Co Pub & Restaurant - Lakeside - (Rohnert Park)
- Bird Street Brewing - (Lemoore)
- Black Hammer Brewing - (San Francisco)
- Brewbakers Brewing Co - (Visalia)
- Brewery in Planning - Los Angeles - (Los Angeles)
- Brewery in Planning - San Diego - (San Diego)
- Brewery Twenty Five - (Hollister)
- Camino Brewing Co LLC - (San Jose)
- Chula Vista Brewery - (Chula Vista)
- Cloverdale Ale Company's Ruth McGowan's Brewpub - (Cloverdale)
- Craftsman Brewing Co - (Pasadena)
- Devil's Potion Brewing Company LLC - (Escondido)
- Dry River Brewing - (Los Angeles)
- Eight Bridges Brewing - (Livermore)
- Eppig Brewing - (San Diego)

3 Fetch() methode

📄 [pokemon.html – script/pokemon.js - https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json](https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json)

3.1 Wat is een promise?

De fetch() methode werkt iets anders dan we gewoon zijn. Deze functie werkt met een *Promise*.

Een normale methode of functie werkt synchroon. De code binnen een functie moet volledig doorlopen zijn voordat de andere programmeercode na de oproep uitgevoerd kan worden.

Een methode die met *Promises* werkt –zoals de fetch() methode- werkt asynchroon. Op deze manier "bevriest" de browser niet tot er een response van de server is teruggekomen.

Een methode die met *Promises* werkt "belooft" aan de code dat hij een succeeded zal versturen zodra hij fulfilled (klaar) is.

In de code wordt vervolgens het *then blok* uitgevoerd.

Als de functie "fulfilled" is (promise has succeeded) wordt de *then*-methode uitgevoerd.

Als de functie "rejected" is (promise has failed) wordt de *catch*-methode uitgevoerd.

3.1.1 De promise van de fetch-functie

We zien dat de *succeeded promise* van *fetch* een object als parameter heeft die de response van de server voorstelt. We herkennen in het response-object de body en de statuscode (200).

```
const laadData = function() {
    fetch('https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json')
        .then(function(response) {
            console.info('Er is een response teruggekomen van de server');
            console.log(response);
        })
        .catch(function(error) {
            console.error('fout bij verwerken json' + error);
        });
};
```

Wordt uitgevoerd als er een response is Fulfilled.

```
Er is een response teruggekomen van de server
pokemon.js:28
pokemon.js:29
Response {type: "cors", url: "https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json", redirected: false, status: 200, ok: true, ...}
  ↴
  body: (...)
```

3.2 Chained

Wat als we binnen het *then*-blok opnieuw een methode oproepen die ook een *Promise* is (zoals *json()* methode)? Dan moeten we ook voor deze methode een *succeeded Promise* *then*-blok schrijven.

Gelukkig kunnen we van de promises een ketting maken ("chained").

3.2.1 Chained toegepast op fetch-function en json-function

Om het JSON object uit het response object af te zonderen gebruiken we de json()-methode.
Via console.log() zien we in de devTool dat ook deze methode een promise teruggeeft.

```
console.log(response.json())
```

```
▼ Promise ⓘ
  ► __proto__: Promise
  ► [[PromiseStatus]]: "resolved"
  ► [[PromiseValue]]:
    ► pokemon: (151) {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}
  ► __proto__: Object
```

We breiden het vorige voorbeeld uit met een extra then-blok voor de json() methode.
Belangrijk is dat we console.log(response.json()) verwijderen.
Want een promise kan maar 1 maal worden uitgevoerd.

```
const laadData = function() {
  fetch('https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json')
    .then(function(response) {
      console.info('Er is een response teruggekomen van de server');
      /*console.log(response.json());*/
      return response.json();
    })
    .then(function(jsonObject) {
      console.info('json object is aangemaakt');
      console.log(jsonObject);
    })
    .catch(function(error) {
      console.error('fout bij verwerken json' + error);
    });
};
```

Wordt uitgevoerd als het omzetten van
de response naar json gelukt is.

We zien dat de succeeded promise van json() een json-object als parameter heeft. Het voordeel van Chaining is dat de catch zowel fouten in de eerste promise als tweede promise kan oppangen.

```
DOM geladen
Er is een response teruggekomen van de server
json object is aangemaakt

▼ Object ⓘ
  ▼ pokemon: Array(151)
    ▼ [0 .. 99]
      ▷ 0: {id: 1, num: "001", name: "Bulbasaur", img: "http://www.serebii.net/po
      ▷ 1: {id: 2, num: "002", name: "Ivysaur", img: "http://www.serebii.net/poke
      ▷ 2: {id: 3, num: "003", name: "Venusaur", img: "http://www.serebii.net/pok
      ▷ 3: {id: 4, num: "004", name: "Charmander", img: "http://www.serebii.net/po
      ▷ 4: {id: 5, num: "005", name: "Charmeleon", img: "http://www.serebii.net/po
```

Binnen de tweede promise kunnen we het JSON object overlopen zoals we zagen in hoofdstuk 2.4.1 JSON (object) via een string. We kunnen dit pas hier uitwerken omdat - op deze plaats in de code - de data geladen én omgezet is naar een JSON object.

Bij voorkeur noteren we het verwerken in een afzonderlijk deelprobleem (functie) om de leesbaarheid te verhogen.

```

const laadData = function() {
    fetch('https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json')
        .then(function(response) {
            console.info('Er is een response teruggekomen van de server');
            return response.json();
        })
        .then(function(jsonObject) {
            console.info('json object is aangemaakt');

            /* Bij voorkeur noteer je dit in afzonderlijke functie*/
            console.log('verwerken pokemons')
            let arrPokemon = jsonObject.pokemon;
            for (let i = 0; i < arrPokemon.length; i++) {
                console.log(arrPokemon[i].name);
            }
        })
        .catch(function(error) {
            console.error('fout bij verwerken json' + error);
        });
};

```

3.3 De fetch() methode error proof maken

3.3.1 Situatie

Stel dat er iets misgaat bij de communicatie naar de server. Bijvoorbeeld de server is niet bereikbaar of genereert een error. In zo een situatie zal er toch een response komen van de server. Er zal een response zijn, maar echter geen met een statuscode 200 OK.

Maar bijvoorbeeld een 404, 301,... <https://developer.mozilla.org/nl/docs/Web/HTTP/Status>

```

fetch('https://raw.githubusercontent.com/Biuni/PokeonGO-Pokedex/master/pokedexFOUT.json')
...

```

Het probleem is dat de promise ook in zo een situatie wordt uitgevoerd, want er is "een" response teruggekomen op de fetch() aanvraag. Het gevolg is dat er geprobeerd zal worden om van de binnengekomen response een JSON object aan te maken via json().

Dit zal mislukken en zal de ene na de andere error geven in onze javascript code. We zien dat de laatste foutbericht wordt gegenereerd door de 'catch'.

DOM geladen
✖ ▶ GET https://raw.githubusercontent.com/Biuni/PokeonGO-Pokedex/master/pokedexFOUT.json 404 (Not Found)
Er is een response teruggekomen van de server
✖ ▶ fout bij verwerken jsonSyntaxError: Unexpected token : in JSON at position 3

3.3.2 Oplossing 🤓

De oplossing is om bij het eerste then blok te controleren of er een geldige response terugkomt van de server.

Er mag enkel worden verder gegaan indien de statuscode van de response in de 200-299 range ligt. Als dit het geval is, weten we dat er mag worden omgezet naar een json object.

Is het geen geldige response code (bijvoorbeeld 404, 500,...)? Dan creeëren we onze eigen Error, die later wordt opgevangen door de catch blok in de chain.

Als je een controle uitvoert, lees goed de documentatie van je API. Sommige API's sturen ons soms ook een andere HTML statuscode terug. Bijvoorbeeld 302.

```
const laadDataMetExtraControle = function(){
    fetch('https://raw.githubusercontent.com/Biuni/PokeonGO-Pokedex/master/pokedexFOUT.json')
        .then(function(response) {
            if (!response.ok) {
                throw Error('Probleem bij de fetch(). Status Code: ' + response.status);
            } else {
                console.info('Er is een response teruggekomen van de server');
                return response.json();
            }
        })
        .then(function(jsonObject) {
            console.info('json object is aangemaakt');

            /* Bij voorkeur noteer je dit in afzonderlijke functie*/
            console.log('verwerken pokemons')
            let arrPokemon = jsonObject.pokemon;
            for (let i = 0; i < arrPokemon.length; i++) {
                console.log(arrPokemon[i].name);
            }
        })
        .catch(function(error) {
            console.error('fout bij verwerken json' + error);
        });
}

document.addEventListener('DOMContentLoaded', function() {
    console.info('DOM geladen');
    laadDataMetExtraControle();
});
```

```
DOM geladen
✖ Failed to load resource: the server responded with a status raw.githubusercontent.com of 404 (Not Found)
✖ ▶ fout bij verwerken jsonError: Probleem bij de fetch(). Status Code: 404
```

3.4 De arrow functie (ter info)

Omdat de `fetch()` methode een relatief nieuwe javascript functionaliteit is, vind je online veel code voorbeelden die zijn geschreven in de nieuwste javascript arrow-functies syntax.
Arrow functies zijn een andere (kortere) manier om een functie te noteren. In de lessen FSWD gaan we deze schrijfwijze normaal niet gebruiken. Voor de volledigheid tonen we toch een code voorbeeld.

Bij een arrow functie

- Laat je het keyword ***function*** wegvalLEN.
- Schrijf onmiddellijk de ***parameters***.
 - Haakjes zijn niet nodig als de functie maar 1 parameter heeft.
- Noteer een "***fat arrow***".
- Noteer ***de code*** van de functie tussen {}
 - Bestaat de code uit slechts 1 regel? Dan mag je de {} laten wegvalLEN.
(Zoals je ziet bij catch)
- Opgepast, het keyword ***this*** krijgt hier een andere scope in de functie.

```

const laadDataMetExtraControle = function(){
    fetch('https://raw.githubusercontent.com/Biuni/PokeonGO-
Pokedex/master/pokedexFOUT.json')
    .then(function(response) {
        if (!response.ok) {
            throw Error('Probleem bij de fetch(). Status Code: ' +
response.status);
        } else {
            console.info('Er is een response teruggekomen van de server');
            return response.json();
        }
    })
    .then(function(jsonObject) {
        console.info('json object is aangemaakt');

        /* Bij voorkeur noteer je dit in afzonderlijke functie*/
        console.log('verwerken pokemons')
        let arrPokemon = jsonObject.pokemon;
        for (let i = 0; i < arrPokemon.length; i++) {
            console.log(arrPokemon[i].name);
        }
    })
    .catch(function(error) {
        console.error('fout bij verwerken json' + error);
    });
}

document.addEventListener('DOMContentLoaded', function() {
    console.info('DOM geladen');
    laadDataMetExtraControle();
});

```

```

const laadDataMetExtraControle = function(){
    fetch('https://raw.githubusercontent.com/Biuni/PokemonGO-
Pokedex/master/pokedex.json')
    .then(response => {
        if (!response.ok) {
            throw Error('Probleem bij de fetch(). Status Code: ' +
response.status);
        } else {
            console.info('Er is een response teruggekomen van de server');
            return response.json();
        }
    })
    .then( jsonObject => {
        console.info('json object is aangemaakt');

        /* Bij voorkeur noteer je dit in afzonderlijke functie*/
        console.log('verwerken pokemons')
        let arrPokemon = jsonObject.pokemon;
        for (let i = 0; i < arrPokemon.length; i++) {
            console.log(arrPokemon[i].name);
        }
    })
    .catch(error => console.error('fout bij verwerken json' + error));
}

document.addEventListener('DOMContentLoaded', function() {
    console.info('DOM geladen');
    laadDataMetExtraControle();
});

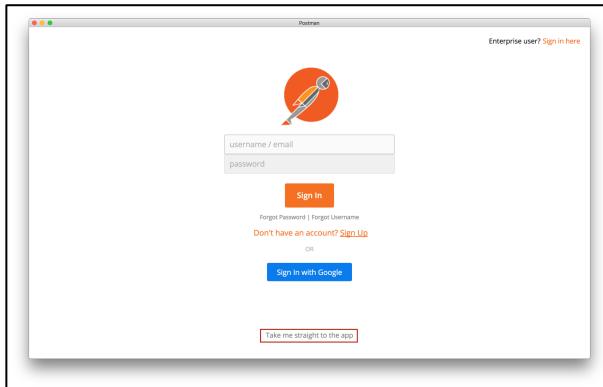
```

4 Labo oefeningen

Tip: voordat je start met het labo: vraag eerst de API key aan voor oefening 4.4 Weather. Het kan even duren voordat de aanvraag aanvaard wordt.

4.1 Postman interpretatie van JSONObject

- Installeer Postman. <https://www.getpostman.com/downloads/>
- Inloggen is niet vereist, kies onderaan voor *Take me straight to the app*.



4.1.1 API: Openlibrary.org

Maak in Postman een GET request naar volgend endpoint:

<https://openlibrary.org/api/books?bibkeys=ISBN:0451526538&format=json>

The screenshot shows the Postman interface with the following details:

- Request method: GET
- URL: https://openlibrary.org/api/books?bibkeys=ISBN:0451526538&format=json
- Params tab selected, showing:
 - bibkeys: ISBN:0451526538
 - format: json
- Body tab selected, showing the response body in JSON format:

```
{<div><table><thead><tr><th>Key</th><th>Value</th><th>Description</th><th>...</th><th>Bulk Edit</th></tr></thead><tbody><tr><td>bibkeys</td><td>ISBN:0451526538</td><td></td><td><input type="checkbox" checked=""></td><td></td></tr><tr><td>format</td><td>json</td><td></td><td><input type="checkbox" checked=""></td><td></td></tr><tr><td>Key</td><td>Value</td><td>Description</td><td><input type="checkbox" checked=""></td><td></td></tr></tbody></table></div>
```
- Status bar: Status: 200 OK Time: 824ms Size: 638 B Save Response

Bespreek en vul volgende vragen aan

Zal deze API een CORS error geven	Ja / Nee / Misschien
Waarom?	
Bestaat het JSONObject uit 1 of 5 objecten	
Van welk gegevenstype is het attribute met key ISBN:0451526538	
Hoe herken je dit gegeventype?	
Van welk gegevenstype is het attribute met key thumbnail_url	
Hoe herken je dit gegevenstype?	

Als het response in javascript zou bewaard zijn in jsonResponse. Hoe vraag je de value van attribute preview_url op?	Console.log(jsonResponse)
---	---

4.1.2 API: swapi.co

Maak in Postman een GET request naar volgend endpoint: <https://swapi.co/api/people/> Dit is de open API met Star Wars data.

Bespreek en vul volgende vragen aan

Zal deze API een CORS error geven	Ja / Nee / Misschien
Waarom?	
Van welk gegevenstype is attribute <i>name</i> ?	
Hoe herken je dit gegeven type?	
Van welk gegevenstype is attribute <i>films</i> ?	
Hoe herken je dit gegevenstype?	

4.1.3 API: opentdb.info

Maak in Postman een GET request naar volgend endpoint: <https://opentdb.com/api.php?amount=5>

Deze API genereert 5 quizvragen, met bijhorende antwoorden.

Bespreek en vul volgende vragen aan

Zal Rest API een CORS error geven	Ja / Nee / Misschien
Waarom?	
Van welk gegevenstype is het attribute met key results	
Als het response in javascript zou bewaard zijn als jsonResponse	let mogelijkeVragen; mogelijkeVragen = jsonResponse.....
Vervolledig de javascript code om de 5 vragen onder elkaar af te printen	for (let vraagmogelijkeVragen){ console.log(.....) }

4.1.4 API: tvmaze.com

Maak in Postman een GET request naar volgend endpoint:

<http://api.tvmaze.com/singlesearch/shows?q=Homeland&embed=episodes>

Deze API toont de info over een bepaalde tvserie.

Bespreek en vul volgende vragen aan

Zal deze API een CORS error geven	Ja / Nee / Misschien
Waarom?	
Van welk gegevenstype is het attribute met key runtime	
Hoe herken je dit?	
Wanneer wordt Homeland uitgezonden? Waar heb je dit teruggevonden?	
Wat is het gegevenstype van het attribute met key webChannel, waarom zou dit zijn?	

Zoek de samenvatting van episode 5 van seizoen 1? Waar heb je deze gevonden.	
Zoek de afbeelding die bij deze aflevering hoort. Zoekt deze met medium kwaliteit. Hoe verwijst je naar deze waarde?	

4.2 Chrome devTool toepassing

- ☰ Open labo/devtool in VS Code –
- ☰ API: <http://api.tvmaze.com/singlesearch/shows?q=Homeland&embed=episodes>



4.2.1 Opdracht

Vul de javascript code in serie.js verder aan zodat we volgende output krijgen in de console van de Chrome devTool.

We gebruiken het JSON object dat we terugkrijgen van het endpoint van tvmaze. Die zal alle informatie bevatten over de TVserie Homeland.

```
De naam van de serie is Homeland
Het uur waarop het wordt uitgezonden is 21:00
Volgende genres
- Drama
- Thriller
- Espionage
Overzicht episodes
s 1 e 1 - Pilot
s 1 e 2 - Grace
s 1 e 3 - Clean Skin
s 1 e 4 - Semper I
s 1 e 5 - Blind Spot
s 1 e 6 - The Good Soldier
s 1 e 7 - The Weekend
s 1 e 8 - Achilles Heel
s 1 e 9 - Crossfire
```

4.2.2 Stappenplan

1. Open het endpoint in Postman.
 - a. Controleer of de header aanwezig is.
 - b. Verken het JSON object. Object of Array? Gegevenstypes?
2. Controleer de communicatie tussen de API en je webpagina
 - a. Is serie.js gekoppeld aan serie.html?
 - b. Wordt *DOMContentLoaded* uitgevoerd? Via console.log().
 - c. Worden de *fetch()*-then-promise en *json()*-then-promise uitgevoerd? Via console.log() controleren we dit.
3. Welke data kan er worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (let of const)
4. Is er data die in een Array staat? Hiervoor gebruiken we een lus-structuur. (*for()* of *for of()*)

4.3 Afvalcontainers van Antwerpen

Open labo/afvalcontainer in VS Code

API:

https://opendata.arcgis.com/datasets/413c00cfda8743fbb94ce7e7e67d67c7_49.geojson

```
afvalcontainer
script
app_antwerpen.js
style
antwerpen.css
antwerpen.html
```

4.3.1 Opdracht

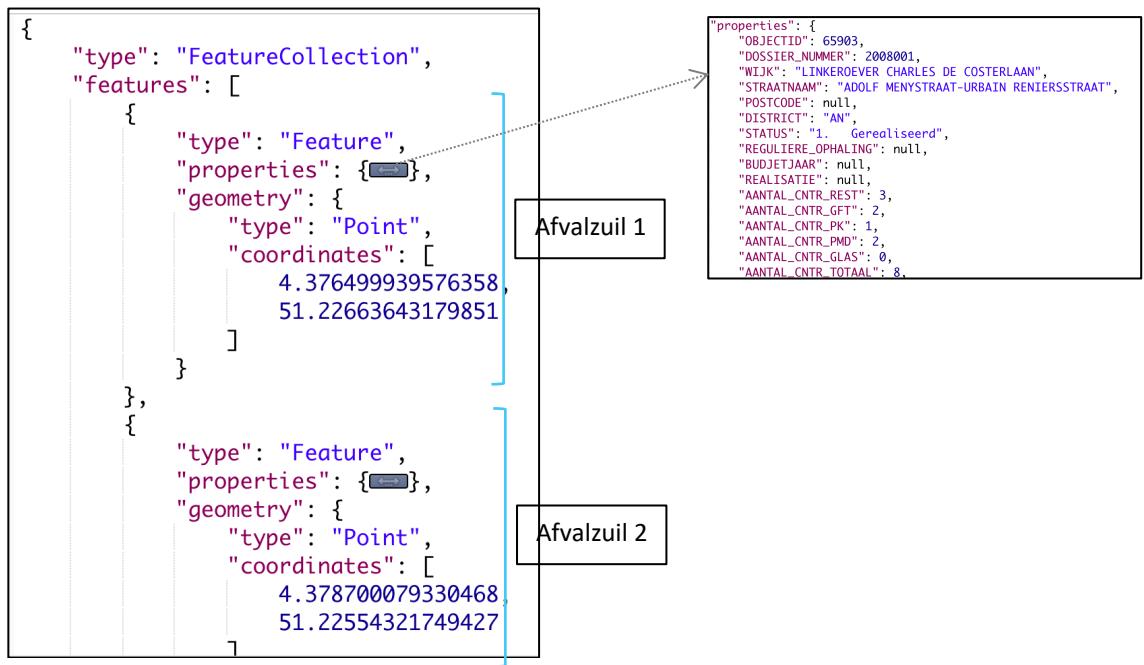
Toon een overzicht van alle afvalzuilen die aanwezig zijn in Antwerpen. Per 'afvalzuil locatie' tonen we hoeveel GFT, GLAS,... containers er beschikbaar zijn.

Afvalzuilen in Antwerpen				
ADOLF MENYESTRAAT-URBAIN RENIERSSTRAAT				
GFT 2	GLAS ---	PMD 2	REST 3	PAPIER 1
HALEWIJNLAAN				
GFT 1	GLAS 1	PMD 1	REST 2	PAPIER 1
HALEWIJNLAAN				
GFT 1	GLAS ---	PMD 1	REST 2	PAPIER 1
WILLEM ELSSCHOTSTRAAT				
GFT 1	GLAS ---	PMD 1	REST 2	PAPIER 1
CHARLES DE COSTERLAAN				
GFT 1	GLAS ---	PMD 1	REST 2	PAPIER 1
ALBERT NOLET LAAN				

We krijgen reeds een statisch HTML document dat gekoppeld is aan een css bestand. Pas dit bestand verder aan, zodat de gegevens uit de API worden geladen.

4.3.2 Stappenplan

1. Open het endpoint in Postman.
 - a. Controleer of de header aanwezig is.
 - b. Verken het JSONobject. Object of Array? Gegevenstypes?
2. Controleer de communicatie tussen de API en de webpagina
 - a. Koppel app_antwerpen.js aan antwerpen.html.
 - b. Schrijf de `addEventListener DOMContentLoaded?` Controleer via `console.info()` of deze wordt uitgevoerd.
 - c. Schrijf de `fetch()-then-promise` en `json()-then-promise` in een afzonderlijke functie `laadDataAfvalContainers()`. Controleer via `console.info()` of deze wordt uitgevoerd.
 - d. Schrijf een afzonderlijke functie `verwerkAfvalContainers ()` om de HTML code op te bouwen.
3. Is er data die kan worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (let of const)
 - a. Ja/Neen
 - b. ~~Met welke html element komt dit overeen? Spreek deze aan via `document.querySelector('')` en pas de `innerHTML` aan.~~
4. Is er data die in een Array staat? Hier voor gebruiken we een lus-structuur. (`for()` of `for of()`).
 - a. Ja/Neen -> `features[...,...,...,...]`



- b. Met welk html-blok zal dit overeenkomen? Verwijder de inhoud binnen het HTML bestand, en spreek het parent-html-element aan via `document.querySelector('')`

```

<body>
  <h1>Afvalzuilen in Antwerpen</h1>
  <div class="js-placeholder">
    <article class="c-locatie">...
    </article>
    <article class="c-locatie">...
    </article>
    <article class="c-locatie">...
    </article>
    <article class="c-locatie">...
    </article>
  </div>
</body>

```



```

<body>
  <h1>Afvalzuilen in Antwerpen</h1>
  <div class="js-placeholder">
    ...
    containers worden opgehaald ...
  </div>
</body>

```

- c. Haal binnen de `for`-lus de waarden op die je nodig hebt.

- Bewaar deze in een variabele. Toon deze eventueel via `console.log()`
- Bouw de html op voor 1 afvalzuil (in dezelfde for-lus) en voeg deze toe aan de `innerHTML` van het parent-element. Gebruik hiervoor `(+=)`

4.3.3 Afwerking

Maak een extra functie `zeroToStreep(waarde)`

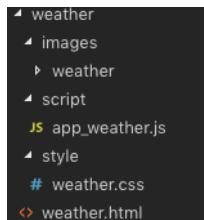
- Die de binnengekomende parameter controleert.
- Een '---' teruggeeft, bij een parameter die gelijk is aan 0.
- Of de waarde van de parameter teruggeeft, als de waarde groter is dan 0.
- Gebruik deze functie in `verwerkAfvalContainers()`.

Afvalzuilen in Antwerpen				
ADOLF MENYSTRAAT-URBAIN RENIERSSTRAAT				
GFT 2	GLAS ---	PMD 2	REST 3	PAPIER 1
HALEWIJNLAAN				
GFT 1	GLAS 1	PMD 1	REST 2	PAPIER 1
HALEWIJNLAAN				
GFT 1	GLAS ---	PMD 1	REST 2	PAPIER 1

4.4 Weather

Open labo/weather in VS Code

API: http://api.openweathermap.org/data/2.5/forecast?q=kortrijk,BE&appid=*****&units=metric&lang=nl



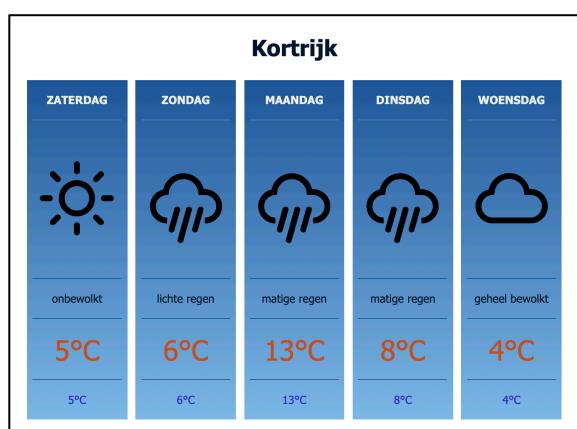
4.4.1 Vooraf

Om deze API te gebruiken hebben we een appid-code nodig van openweathermap.com. Deze kan je gratis aanmaken.

- Surf naar <https://home.openweathermap.org>
- Registreer je.
- Nadat je bent ingelogd ga naar API keys.
- Deze key zal je als waarde gebruiken in je endpoint.

4.4.2 Opdracht

Toon de vijfdaagse weersvoorspelling. Toon de dag, een icoon die de weerstoestand voorstelt, de weerstoestand, de max en minimum temperatuur. Pas eventueel de parameter in je endpoint aan om het weer in een andere stad op te vragen.



4.4.3 Stappenplan

1. Open het endpoint in Postman.
 - a. Controleer of de header aanwezig is.
 - b. Verken het JSONobject. Object of Array? Gegevenstypes? We verwachten een voorspelling van 5 dagen, maar toch zien we er veel meer? Hoe komt dit? Hoe zullen we dit straks oplossen om toch maar 5 voorspellingen te tonen? Waar staat de locatie? Hoe wordt de datum weergegeven?
2. Controleer de communicatie tussen de API en de webpagina
 - a. Koppel app_weather.js aan weather.html.
 - b. Schrijf de addEventListener DOMContentLoaded? Controleer via console.info() of deze wordt uitgevoerd.
 - c. Schrijf de fetch()-then-promise en json()-then-promise in een afzonderlijke functie laadWeatherInfo(). Controleer via console.info() of deze wordt uitgevoerd.
 - d. Schrijf een afzonderlijke functie verwerkWeer() om de HTML code op te bouwen eenmaal het omzetten naar een JSON object is gelukt.
3. Is er data die kan worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (let of const)
 - a. Met welk html element komt dit overeen? Spreek dit element aan via document.querySelector('') en pas de innerHTML aan.
4. Is er data die in een Array staat? Hier voor heb je een lus-structuur nodig. (for() of for of()).
 - a. Met welk html-blok zal dit overeenkomen? Verwijder de inhoud en spreek de het parent-html-element aan via document.querySelector('')
 - b. Haal binnen de for-lus de waarden op die je nodig hebt; bewaar deze in een variabele of constante. Toon deze eventueel via console.log()
 - c. Bouw de HTML op (in dezelfde for-lus) en voeg deze toe aan de innerHTML van het parent-element.

4.4.4 Tips voor deze oefening

- **Voorspellingen**

De voorspellingen worden per 3 uur getoond. Hoe kunnen we de for-lus zodanig schrijven dat we maar 1 voorspelling per dag opvragen?

- **Weekdag**

De datum (dt) wordt weergegeven als integer volgens het UTC formaat in het JSON object. Om van deze integer een datum te maken moeten we de waarde vermenigvuldigen met 1000 en meegeven aan de constructor van de Date class.

De functie getDay() geeft een getal terug dat het dagnummer voorstelt. (0=maandag, 1 = dinsdag, 2= woensdag, 3 =....).

```
const element = arrVoorspelling[index];
const datumUtc = element.dt;
const dag = new Date(datumUtc * 1000);
const dagnummer = dag.getDay();
```

- **Afbeelding**

Maak een functie weatherCodeToImage(weercode) die een getal (tussen 100 en 999) als parameter binnen krijgt. Dit getal stelt een weerfenomeen voor.

De functie geeft de bijhorende afbeelding terug als string.

Om het eerste karakter van een string af te zonderen, gebruiken we de substring() functie.

```
let codeString = code.toString();
let eersteDigit = codeString.substring(0, 1);
```

weercode	afbeelding
----------	------------

2xx	wi-thunderstorm.svg
3xx	wi-sprinkle.svg
5xx	wi-rain.svg
6xx	wi-snow.svg
7xx	wi-fog.svg
8xx	wi-cloud.svg
800	wi-day-sunny.svg
801	wi-day-sunny-overcast.svg
900	wi-thunderstorm.svg

4.5 Thuisopdracht: Chrome devTool – Deelauto's

- █ Open de map labo/devtool_thuis_deelauto in VS Code
- █ API: <https://datatank.stad.gent/4/mobiliteit/deelwagenspartago>

4.5.1 Opdracht

Je zal volgend endpoint aanspreken: <https://datatank.stad.gent/4/mobiliteit/deelwagenspartago>
 Deze API bevat alle deelauto's van Partago in Gent.

- Verken in Postman de response dat je terug krijgt. Je zal opmerken dat het geen object is maar onmiddellijk een "Array van Objecten".
 - Hoe zie je dit? Welk effect zal dit hebben voor de code?
- Vervolledig de code in deelauto.js om volgende output te krijgen in de console.

```
**** Volgende auto's zijn beschikbaar
Partago Achiel rijdt op electric
Partago Alfa rijdt op electric
Partago Bowie rijdt op electric
Partago Bono rijdt op electric
Partago Gaston rijdt op electric
Partago Leo rijdt op electric
Partago Cesar rijdt op electric
Partago Mees rijdt op electric
Partago Curie rijdt op electric
Partago Dalton rijdt op electric
```

4.5.2 Stappenplan

1. Open het endpoint in Postman.
 - a. Controleer of de header aanwezig is.
 - b. Verken het JSONObject. Object of Array? Gegevenstypes?
2. Controleer de communicatie tussen de API en de webpagina
 - a. Is *.js gekoppeld aan *.html?
 - b. Wordt DOMContentLoaded uitgevoerd? Controleer via console.log().
 - c. Worden of de *fetch()*-then-promise en *json()*-then-promise uitgevoerd? Controleer via console.log().
3. Welke data kan er worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (*let* of *const*)
 - a. Met welk html element komt dit overeen? Spreek dit element aan via *document.querySelector('')* en pas de *innerHTML* aan.
4. Is er data die in een Array staat? Hiervoor heb je een lus-structuur nodig. (*for()* of *for of()*)
 - a. Met welk html element komt dit overeen? Spreek dit element aan via *document.querySelector('')* en pas de *innerHTML* aan.

4.6 Thuisopdracht: Chrome devTool – Open Library

Open de map labo/devtool_thuis_library in VS Code

API: <http://openlibrary.org/search.json?q=keuken>

4.6.1 Opdracht

Je zal volgend endpoint aanspreken: <http://openlibrary.org/search.json?q=keuken>

Deze API geeft een boekenlijst terug die voldoet aan de query "keuken".

- Verken in Postman de response die je terugkrijgt.
- Vervolledig de code in library.js om volgende output te krijgen in je console.

```
aantal gevonden boeken 43
Title en ondertitel Wij zijn 17
Geschreven door Johan van der Keuken
Boek gaat over
-> Accessible book
*****
Title en ondertitel Huis, tuin en keuken
Geschreven door MaartenJan Hoekstra
Boek gaat over
-> Housing
-> Architecture
-> Dutch
-> Interior decoration
-> Terminology
-> History
*****
Title en ondertitel Herman Hertzberger
Geschreven door Herman Hertzberger, Johan van der Keuken
Boek gaat over
-> Exhibitions
-> Architecture
*****
```

4.6.2 Stappenplan

- Open het endpoint in Postman.
 - Controleer of de header aanwezig is.
 - Verken het JSON object. Object of Array? Gegevenstypes?
- Controleer de communicatie tussen de API en je webpagina
 - Is *.js gekoppeld aan *.html?
 - Wordt *DOMContentLoaded* uitgevoerd? Controleer via `console.log()`.
 - Worden de `fetch()`-then-`promise` en `json()`-then-`promise` uitgevoerd? Controleer via `console.log()`.
- Welke data kan er worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (let of const)
 - Met welk html element komt dit overeen? Spreek dit element aan via `document.querySelector('')` en pas de `innerHTML` aan.
- Is er data die in een Array staat? Hiervoor heb je een lus-structuur nodig. (`for()` of `for of()`)
 - Met welk html element komt dit overeen? Spreek dit element aan via `document.querySelector('')` en pas de `innerHTML` aan.

4.6.3 Tip voor deze oefening

Je zal zien dat bij sommige boeken de Array met onderwerpen ontbreekt. (De ontwerpers van deze API hadden beter een lege Array teruggestuurd.)

Om te vermijden dat je een error krijgt wanneer je de Array probeert te overlopen, kan je eerst een controle doen of deze Array wel bestaat.

```
if (boek.subject){  
    ...  
}else{  
    console.log('--- GEEN ONDERWERPEN BESCHIKBAAR ---')  
}
```

```
*****  
Title en ondertitel Krekels in de keuken  
Geschreven door Renée van Riessen  
Boek gaat over  
--- GEEN ONDERWERPEN BESCHIKBAAR ---  
*****
```

4.7 Thuisopdracht: Uitbreiding weather

4.7.1 Opdracht

Pas de html pagina en javascript code zodanig aan dat je ook de windrichting en windkracht toont.

- **Windrichting**

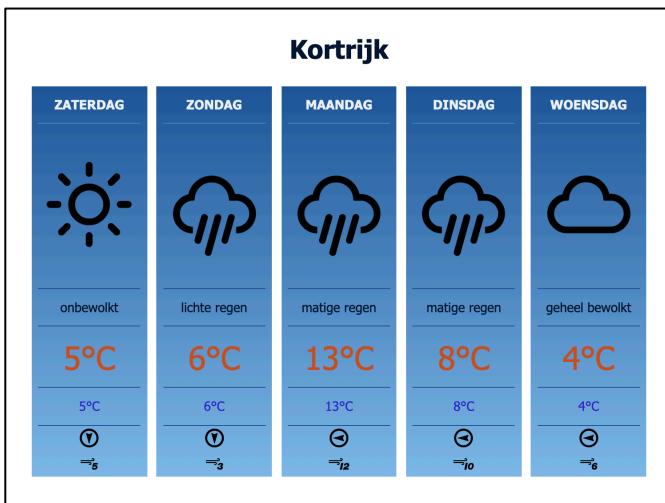
Graden	Richting (afbeelding)
≥ 315 en < 45	N
≥ 45 en < 135	O
≥ 135 en < 225	Z
≥ 225 en < 315	W

- **Windkracht**

Rond de windkracht af naar het dichtstbijzijnde geheel getal.

Je ziet dat je nog een extra div element met twee afbeeldingen toevoegt. (.c-forecast__wind)

```
▼<div class="c-forecast">  
  <div class="c-forecast__datum">maandag</div>  
  ▶<div class="c-forecast__symbol">..</div>  
  <div class="c-forecast__uitleg">  
    matige regen  
  </div>  
  <div class="c-forecast__max">13°C</div>  
  <div class="c-forecast__min">13°C</div>  
  <!--uitbreiding-->  
  ▼<div class="c-forecast__wind">  
      
      
  </div>  
  <!--end uitbreiding-->  
</div>
```



4.8 Thuisopdracht: Bluebike

Open labo/bluebike_thuis in VS Code

API: <https://datatank.stad.gent/4/mobiliteit/bluebikedelfietsensintpieters>

```

├── bluebike_thuis
    ├── images
    │   ├── bluebike
    │   │   └── cover.jpg
    ├── script
    │   └── app_bluebike.js
    ├── style
    │   └── bluebike.css
    └── bluebike.html

```

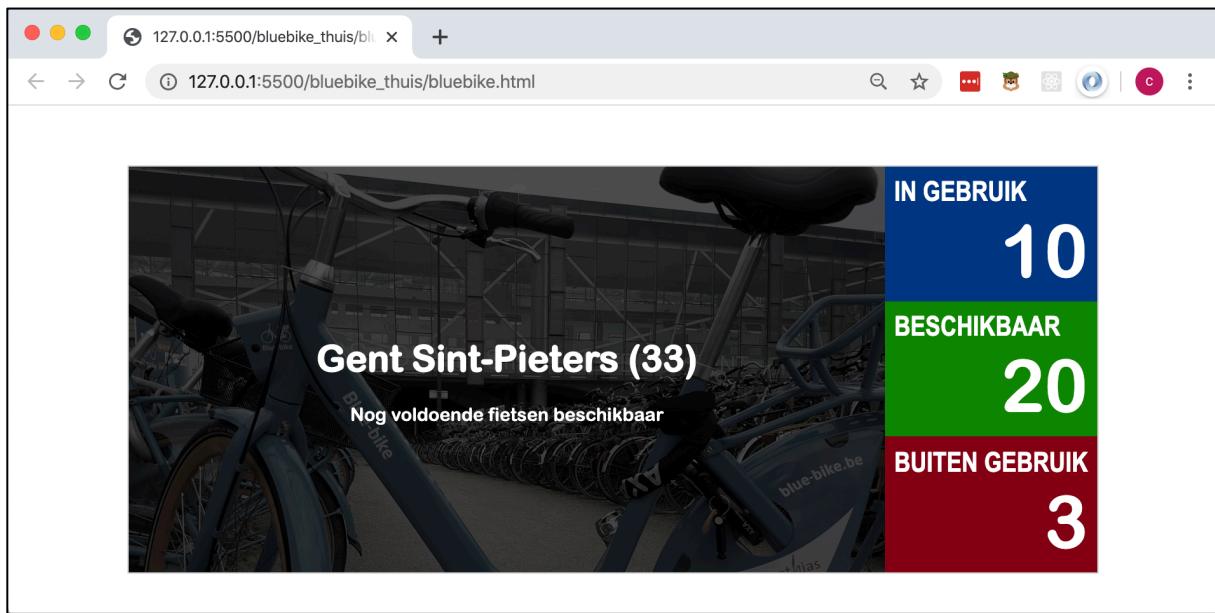
4.8.1 Opdracht

Toon het aantal beschikbare deelfietsen aan het station. Zorg dat er een boodschap verschijnt als er meer fietsen in gebruik zijn dan er nog beschikbaar zijn.

"Haast je, meer dan de helft is reeds in gebruik!" of "Nog voldoende fietsen beschikbaar"

4.8.2 Stappenplan

1. Open het endpoint in Postman.
 - a. Controleer of de header aanwezig is.
 - b. Verken het JSONObject. Object of Array? Gegevenstypes?
2. Controleer de communicatie tussen de API en je webpagina
 - a. Is *.js gekoppeld aan *.html?
 - b. Wordt *DOMContentLoaded* uitgevoerd? Controleer via `console.log()`.
 - c. Worden de `fetch()`-then-`promise` en `json()`-then-`promise` uitgevoerd? Controleer via `console.log()`.
3. Welke data kan er worden ingelezen zonder lus-structuur? Lees eerst deze in en bewaar deze in een variabele (let of const)
 - a. Met welk HTML element komt dit overeen? Spreek dit element aan via `document.querySelector('')` en pas de `innerHTML` aan.
4. Is er data die in een Array staat? Hiervoor heb je een lus-structuur nodig. (`for()` of `for of()`)
 - a. Met welk HTML element komt dit overeen? Spreek dit element aan via `document.querySelector('')` en pas de `innerHTML` aan.



Gebruik in deze oefening bij voorkeur de verschillende .js- klassen in de querySelectors. Op deze manier moet je maar kleine stukjes toevoegen via innerHTML.

```
<div class="c-place">
  <div class="c-place__cover">
    <h1 class="c-place__city js-city">...</h1>
    <span class="c-place__message js-message">...</span>
  </div>
  <div class="c-place__bikes">
    <div class="c-place__status c-place__status--in-use">
      <h2 class="c-place__label">In gebruik</h2>
      <div class="c-place__amount js-amount-in-use">...</div>
    </div>
    <div class="c-place__status c-place__status--available">
      <h2 class="c-place__label">Beschikbaar</h2>
      <div class="c-place__amount js-amount-available">...</div>
    </div>
    <div class="c-place__status c-place__status--out-of-use">
      <h2 class="c-place__label">Buiten gebruik</h2>
      <div class="c-place__amount js-amount-out-of-use">...</div>
    </div>
  </div>
</div>
```

```
const htmlBoodschap = document.querySelector('.js-message');
const htmlBeschikbaar = document.querySelector('.js-amount-available');
const htmlGebruik = document.querySelector('.js-amount-in-use');
const htmlDefect = document.querySelector('.js-amount-out-of-use');
const htmlLocatie = document.querySelector('.js-city');
```



howest
hogeschool