# Week 11, 12 en 13

## Javascript Deel 6

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

# Inleiding JSON

# Wat is JSON?

- JavaScript Object Notation
- Tekst
- Een zelfbeschrijvende syntax voor het sorteren/uitwisselen van data

*JSON maakt het mogelijk om eenvoudig met data als Javascript objecten te werken!*

# JSON Syntax

- JSON wordt op volgende manier geschreven:
  - Data wordt altijd geschreven in naam-waarde-paren
  - Data wordt gescheiden door komma’s
  - Accolades staan rond de objecten
  - Vierkante haakjes staan rond de arrays

# JSON Syntax

- JSON data als naam-waarde-paren:
  `“voornaam”:“Rik”`

- JSON object binnen accolades en data gescheiden door komma’s:
  `{“voornaam”:“Rik”, “leeftijd”:21, “stad”:“Genk”}`

# JSON Syntax

- JSON arrays worden geschreven binnen vierkante haakjes:

```
“studenten” = [
{“voornaam”:“Rik”, “leeftijd”:21, “stad”:“Genk”},
{“voornaam”:“Anne”, “leeftijd”:19, “stad”:”Hasselt”}
]
```

# Data verzenden

- Wanneer je data hebt opgeslagen in een Javascript object, kan je dit object converteren naar JSON en naar de server verzenden *(gaan we in deze cursus niet doen!).*

```
let persoon = {naam: "Rik", leeftijd: 21, stad: "Genk"};
let persoonJSON = JSON.stringify(persoon);
window.location = "demo_json.php?x=" + persoonJSON;
```

# Data ontvangen

- Wanneer je data ontvangt in een JSON-formaat, kan je deze converteren naar een Javascript object.

```
let persoonJSON = '{"naam":"Rik", "leeftijd":21, "stad":"Genk"}';
let persoon = JSON.parse(persoonJSON);
document.getElementById("demo").innerHTML = persoon.naam;
```

# Inleiding AJAX

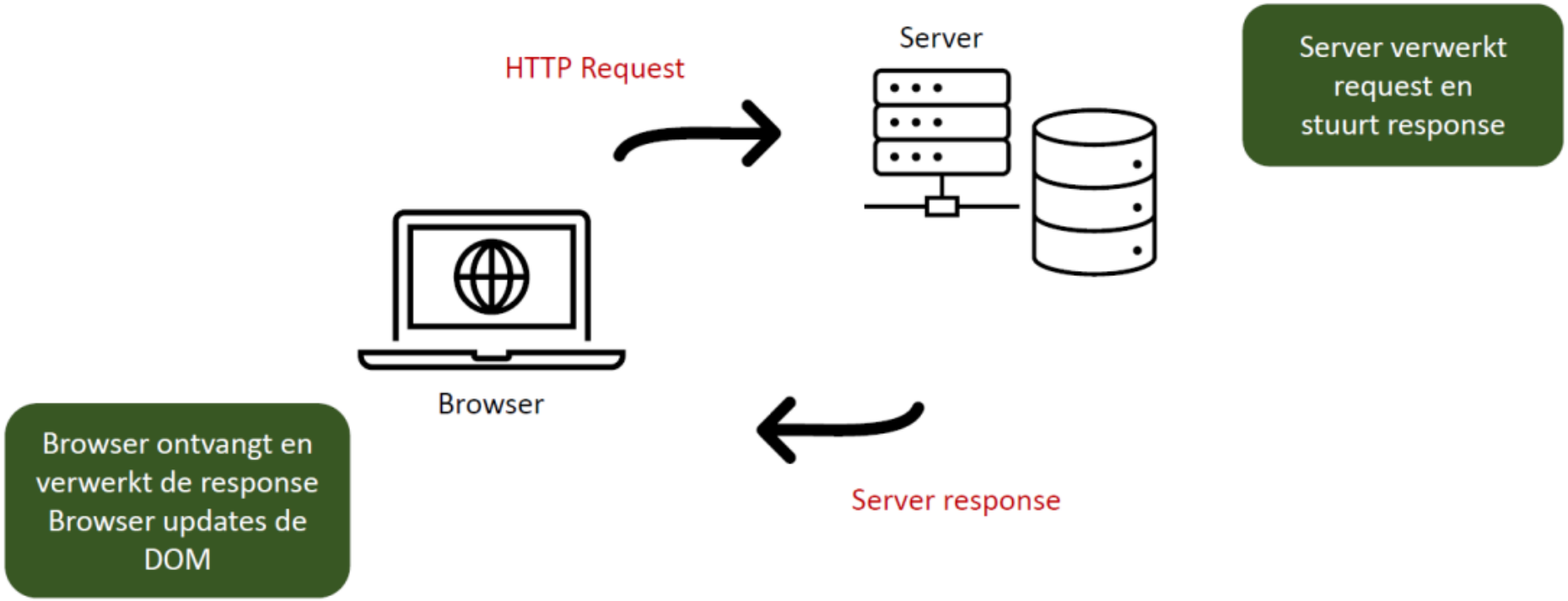
# Wat is AJAX?

- Asynchronous Javascript And XML
- Een manier om interactieve webapplicaties te ontwikkelen door een combinatie van verschillende webtechnologieën te gebruiken:
  - HTML en CSS
  - Javascript
  - DOM
  - XML, JSON, …
  - Fetch, XMLHttpRequest, …

# Wat is AJAX?

- Met AJAX kunnen we, zonder dat er nieuwe pagina-ladingen nodig zijn:
  - data van een server opvragen
  - data van een server ontvangen
  - data naar een server sturen

# Werking AJAX



# HTTP Request

- Met een HTTP request kunnen we de gewenste actie aangeven.

| | |
|---|---|
| **GET** | Data ontvangen, data lezen |
| **POST** | Data versturen, data creëren |
| **PUT** | Data uploaden, data wijzigen |
| **DELETE** | Data verwijderen |
| **…** | … |

# Het XMLHttpRequest object

- Met het XMLHttpRequest object kunnen we op een gemakkelijke manier informatie uit een URL ophalen zonder de gehele pagina te herladen. De webpagina kan een gedeelte van de pagina bijwerken zonder dat de gebruiker daar last van heeft.

- Deze manier gaan we niet gebruiken in deze cursus!

# Wat gaan we wel gebruiken? Waarom?!

| XMLHttpRequest | Fetch |
| --- | --- |
| Ondersteuning door moderne browsers<br>Bestaat reeds geruime tijd | Makkelijker in gebruik en krachtiger<br>Maakt gebruik van ‘Promises’<br>Toegang overheen domeinen mogelijk. De webpagina en de bestanden kunnen op verschillende servers staan, |
| Slordiger en ingewikkelder<br>Geen toegang meer overheen domeinen door diverse moderne browsers, de webpagina en de bestanden moeten op dezelfde server staan | Relatief nieuw<br>Comptabiliteit in sommige browsers nog problematisch |

# Fetch API

# Werking Fetch API en promises

- De Fetch API werkt met *promises*.
- Een promis is …
  - een object dat de uiteindelijke voltooiing of mislukking van een asynchroon event vertegenwoordigt.
  - een manier om met een asynchroon event om te gaan.
  - een soort van ‘placeholder’ of ‘proxy’ voor de response dat we asynchroon (in de toekomst) gaan krijgen.

# Werking Fetch API en promises

- Een promise heeft altijd één van deze drie statussen:
  - Pending, de initiële status, de actie is niet fulfilled én niet rejected.
  - Fulfilled, de actie is succesvol voltooid.
  - Rejected, de actie is mislukt.

- Een ‘pending’ promise kan zowel de status ‘fulfilled’ als ‘rejected’ krijgen. Een promise roept, wanneer ‘fulfilled’, de then-methode met een response-object op.

# GET – Data ophalen

```
fetch('URL')                          // pad van de bron ingeven

.then(function(response) {            // Response (data) in JSON-formaat ophalen.
      return response.json()
})
.then(function(data) {                // Response (data) in de console weergeven.
      console.log(data)
})
.catch(function(error) {              // Eventuele error in console weergeven.
      console.log(error)
})
```

# GET – Data ophalen (vereenvoudiging)

```
fetch('URL')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.log(error))
```

# Voorbeelden

Voorbeeld 1 – Random User ophalen en weergeven in console

Voorbeeld 2 – Random User ophalen en weergeven op webpagina

# POST – Data versturen

```
fetch('URL', {
       method: 'POST’,                  // Geef de methode op die we gebruiken.
       headers: {                       // Voeg de HTTP Headers toe.
       "Content-type": "application/json; charset=UTF-8“
       },
       body: JSON.stringify({           // Voeg de body/inhoud toe in JSON-formaat.
           inhoud: "Dit is de inhoud“
     })
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.log(error))
```

# Voorbeelden

Voorbeeld 1 – Post met vaste titel en inhoud

Voorbeeld 2 – Post met titel en inhoud ingegeven door bezoeker webpagina