

The background of the slide features a photograph of a modern building with a facade of large, light-colored concrete panels. Some panels are recessed, creating a textured, three-dimensional effect. A person with long brown hair, wearing a green jacket, blue jeans, and a red backpack, is walking from left to right in the lower half of the image. There are some green plants in the bottom left corner. Two white rectangular boxes are placed over the image: one in the top left and one in the middle left.

Webapplicaties II

Hoofdstuk 08: Ajax – fetch API

Inhoud

- Ajax
- Ajax en callback
- Ajax en promises
- Fetch API

08: Ajax – fetch API

Asynchronous **J**avaScript **a**nd **X**ML

AJAX (spreek uit: eidzæks)

- **Asynchronous JavaScript And XML (AJAX)** is een term voor het ontwerp van interactieve webpagina's waarin asynchroon gevraagde gegevens worden opgehaald van de webserver. Daardoor hoeven dergelijke pagina's niet in hun geheel ververst te worden.
De term is op 18 februari 2005 door Jesse James Garrett gelanceerd en werd door grote bedrijven als Google en Amazon overgenomen.

AJAX

- Door gebruik te maken van **XMLHttpRequest** hoeft de webpagina niet opnieuw ververst te worden om nieuwe inhoud te krijgen.
- *Google Suggest* stelt bijvoorbeeld bij elke toetsaanslag een nieuwe reeks zoektermen voor zonder dat men de pagina hoeft te herladen. Zo'n pagina is te vergelijken met een applicatie die in de browser draait.

https://nl.wikipedia.org/wiki/Asynchronous_JavaScript_and_XML

AJAX

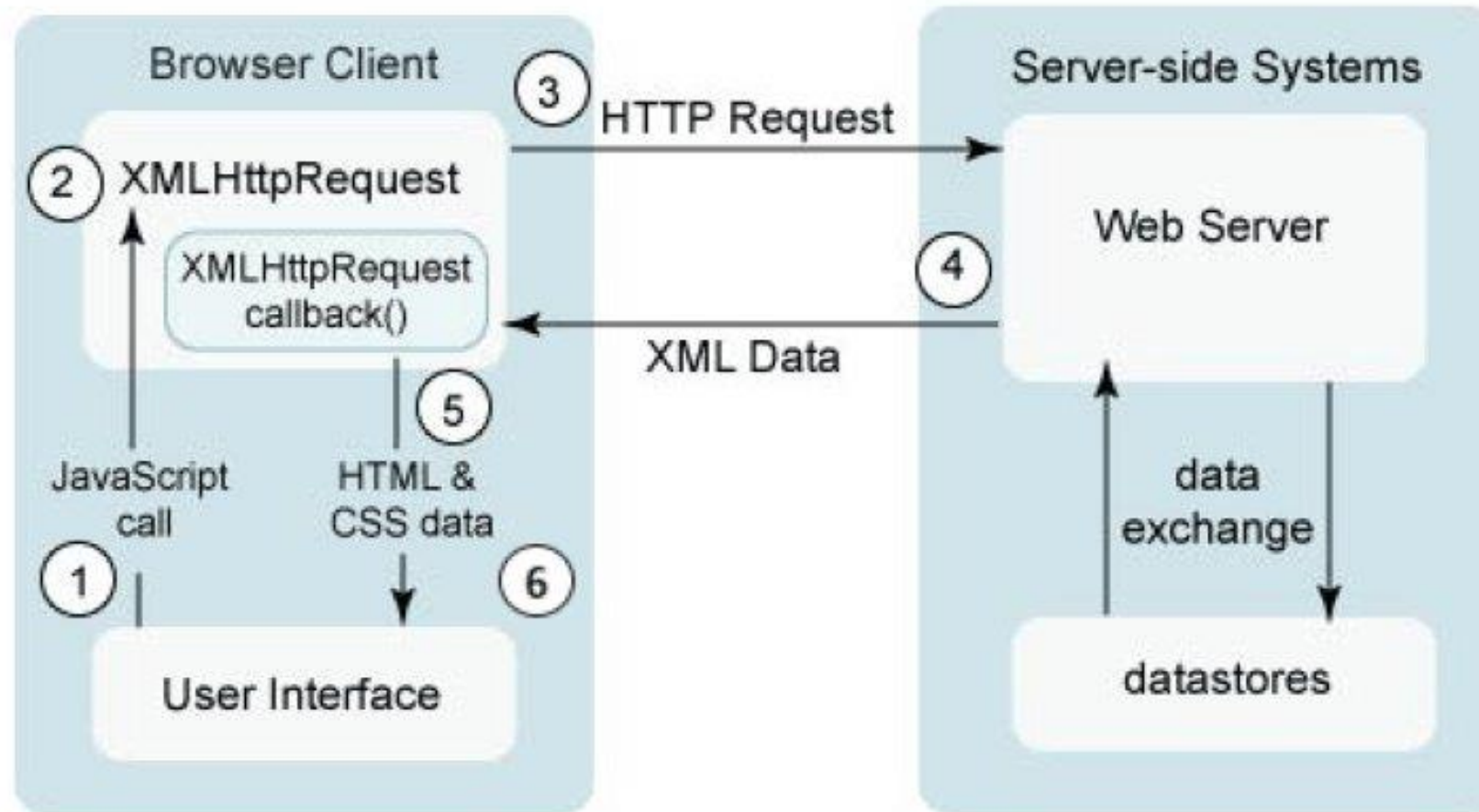
Ajax is een manier om interactieve webapplicaties te ontwikkelen door een combinatie van de volgende technieken te gebruiken:

- **HTML** en **CSS** voor de presentatie volgens de standaarden van het W3C.
- Het **Document Object Model** voor het dynamisch tonen van informatie en voor interactie.
- **XML** voor de opslag, aanpassing en transport van gegevens. In veel gevallen wordt in plaats van XML, **JSON** (JavaScript Object Notation), gewone tekst of een ander formaat gebruikt.
- Het **XMLHttpRequest object** voor asynchrone communicatie met de back-end server.
- **JavaScript** om alles aan elkaar te binden.

AJAX

- Webapplicaties kunnen met Ajax, **asynchroon** data naar de webserver **zenden** en data **ontvangen**, zonder te interfereren met de huidige webpagina.
Data kunnen opgevraagd worden door gebruik te maken van het XML HTTP request object (XHR).
- Ondanks de naam, is het gebruik van XML niet verplicht (**JSON** is een vaker gebruikt alternatief), en de requests hoeven ook niet asynchroon te zijn.
Toen dit echter bedacht en ontwikkeld werd door Microsoft (~1999) was XML dé manier om data uit te wisselen, vandaar.
- Ajax is een belangrijke technologie bij de ontwikkeling van webapplicaties voor mobile devices.

AJAX



Gebruik van Ajax

- Asynchrone requests zorgen ervoor dat meerdere acties op hetzelfde moment kunnen afgehandeld worden (opvragen details en afbeeldingen).
- Requests van browsers worden sneller uitgevoerd.
- Enkel het gedeelte van de pagina dat verandert, wordt aangepast.
- Server verkeer is beperkt – heel voordelig bij applicaties voor mobile devices.



08: Ajax – fetch API

Ajax en callback

Een eenvoudig voorbeeld

- Om Ajax requests te illustreren hebben we een server (backend) nodig waar we data kunnen opvragen.
- We maken gebruik van de grapjes API (https://official-joke-api.appspot.com/random_joke). Er zijn er nog andere, zoals de Star Wars API: <https://swapi.co/>.
- De grapjes API retourneert een grap maar is -vooral- volledig vrij te gebruiken zonder dat we ook authenticatie nodig hebben.

Een eenvoudig voorbeeld

Klikken op de knop moet een grapje ophalen.

```
<div class="container">
  <div class="page-header">
    <h1>Random joke</h1>
  </div>

  <div class="form-group">
    <div class="col-sm-2">
      <input type="button" id="joke" value="Show joke" class="btn btn-default">
    </div>
    <div class="col-sm-5">
      <p id="category"></p>
      <p id="setup"></p>
      <p id="punchline"></p>
    </div>
  </div>
</div>
```

Voorbeeld: tekst-response

- Open Ajax – ajaxText.js.

```
function ajaxRequest(url){
    // creatie van het XHR object
    const request = new XMLHttpRequest();
    // open(Methode, URL, Asynchronous, Username, Password)
    request.open('GET', url);
    // callback functie declareren om de request af te handelen
    request.onreadystatechange = () => {
        // readyState en status worden geëvalueerd
        if (request.readyState === 4 && request.status === 200) {
            console.log(request.responseText);
        }
    };
    // verstuur het verzoek naar de server
    request.send();
}

class JokeApp{
    getData(){
        ajaxRequest('https://official-joke-api.appspot.com/random_joke');
    }
}
```

Voorbeeld: tekst-response

- Open Ajax – ajaxText.js.

```
const init = function(){  
  const jokeApp = new JokeApp();  
  document.getElementById("joke").onclick = () => {  
    jokeApp.getData();  
  };  
}  
  
window.onload = init;
```

```
{"id":200,"type":"general","setup":"What do prisoners use to call each other?","punchline":"Cell phones."}
```

Ajax – XHR object - methodes

- open method:
 - configureert het XHR object.
 - **open(method, url, asynchronous, userName, password)**
 - HTTP request method: GET – POST – HEADER – PUT – DELETE (zie netwerken en webapplicaties 3 en 4)
 - URL: de url waar het request naar verstuurd wordt (het adres). Dit kan een tekst, json, xml bestand zijn. Het kan ook een server-side script zijn (C# - PHP - ...) die het request verwerkt.
 - Asynchronous: boolean die aangeeft of het request al dan niet asynchroon is. Niet verplicht – default true –
 - Username – password: voor eventuele authenticatie. Beide zijn niet verplicht.

Ajax – XHR object - methodes

- **send** method:
 - verstuurt het request naar de server.
 - data kunnen als parameter naar de server gestuurd worden.

AJAX: XHR object - properties

- Properties van het XHR object:
 - **readyState**: bevat de status van het object:

Value	State	Description
0	UNSENT	Client has been created. <code>open()</code> not called yet.
1	OPENED	<code>open()</code> has been called.
2	HEADERS_RECEIVED	<code>send()</code> has been called, and headers and status are available.
3	LOADING	Downloading; <code>responseText</code> holds partial data.
4	DONE	The operation is complete.

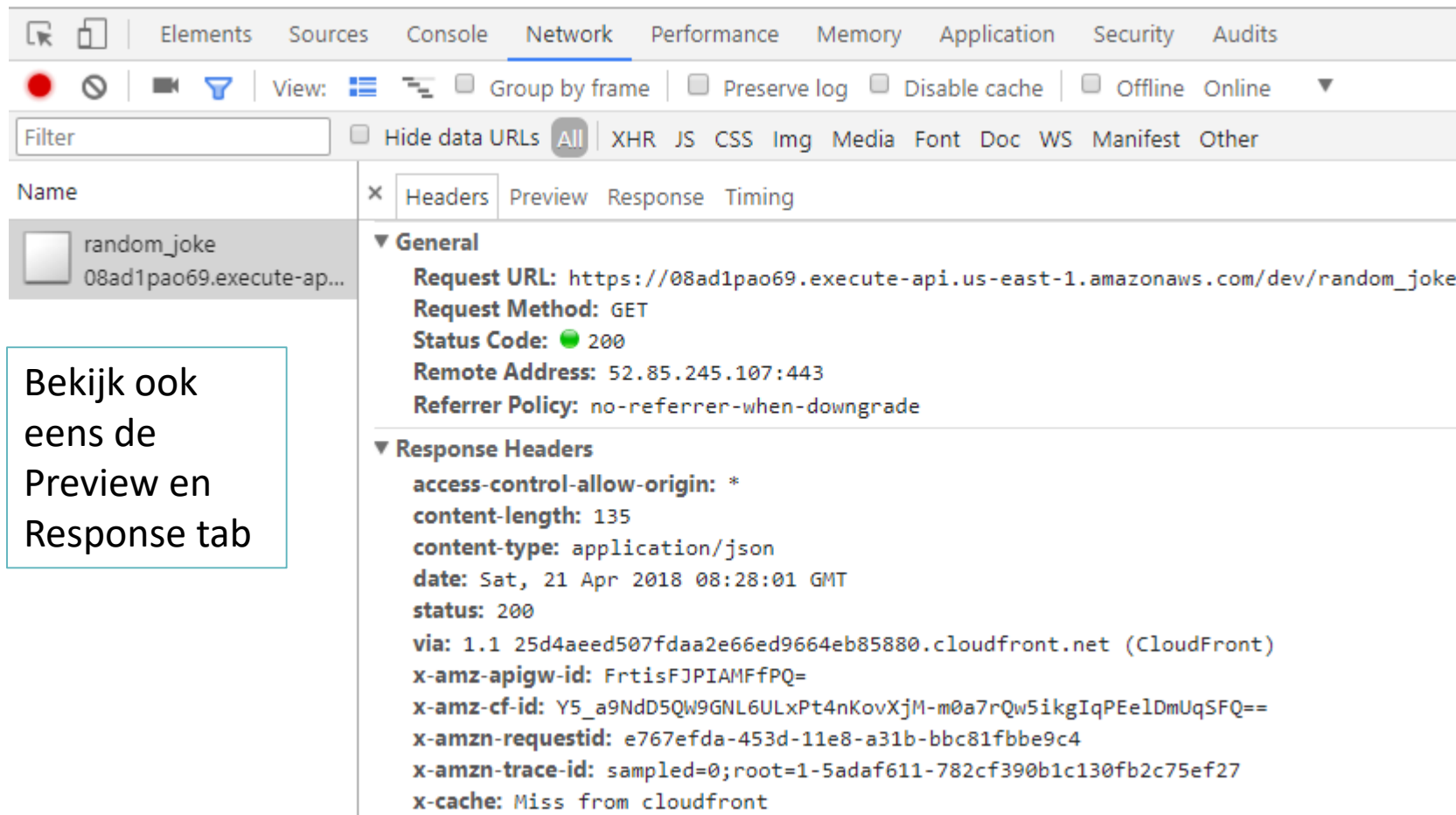
- **status**: geeft de http status die door de server is teruggegeven:
 - 200: "OK"
 - 404: Page not found

AJAX: XHR object - properties

- **onreadystatechange:**
 - koppelt een callback functie
 - deze wordt uitgevoerd elke keer als de readyState van waarde wijzigt.
- **responseText:** de server geeft de data terug als string (kan ook JSON string zijn).
- **responseXML:** de server geeft de data terug als xml.
- Om de verschillende waarden van readyState te bekijken zie voorbeeld XHR.js

Ajax – een eenvoudig voorbeeld

- Met de Chrome developer tools kan je op de Netwerk tab bekijken wat het XMLHttpRequest object heeft opgehaald.



The screenshot shows the Chrome DevTools Network tab. The 'random_joke' request is selected. The 'Headers' tab is active, showing the 'General' section with the following details:

- Request URL:** `https://08ad1pao69.execute-api.us-east-1.amazonaws.com/dev/random_joke`
- Request Method:** `GET`
- Status Code:** `200`
- Remote Address:** `52.85.245.107:443`
- Referrer Policy:** `no-referrer-when-downgrade`

The 'Response Headers' section shows the following details:

- access-control-allow-origin:** `*`
- content-length:** `135`
- content-type:** `application/json`
- date:** `Sat, 21 Apr 2018 08:28:01 GMT`
- status:** `200`
- via:** `1.1 25d4aeed507fdaa2e66ed9664eb85880.cloudfront.net (CloudFront)`
- x-amz-apigw-id:** `FrtisFJPIAMFfPQ=`
- x-amz-cf-id:** `Y5_a9NdD5QW9GNL6ULxPt4nKovXjM-m0a7rQw5ikgIqPEelDmUqSFQ==`
- x-amzn-requestid:** `e767efda-453d-11e8-a31b-bbc81fbbe9c4`
- x-amzn-trace-id:** `sampled=0;root=1-5adaf611-782cf390b1c130fb2c75ef27`
- x-cache:** `Miss from cloudfront`

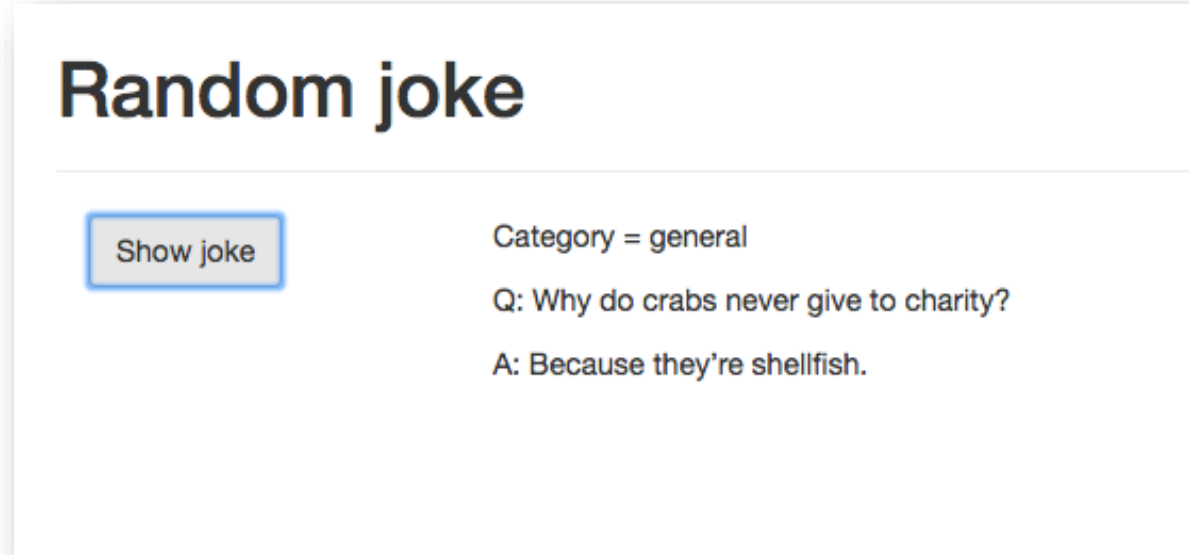
Bekijk ook eens de Preview en Response tab

Voorbeeld – ajaxJSON.js

```
class JokeApp{
  getData(){
    // creatie van het XHR object
    const request = new XMLHttpRequest();
    // open(Methode, URL, Asynchronous, Username, Password)
    request.open('GET', 'https://official-joke-api.appspot.com/random_joke');
    // callback functie declareren om de request af te handelen
    request.onreadystatechange = () => {
      // readyState en status worden geëvalueerd
      if (request.readyState === 4 && request.status === 200) {
        const joke = JSON.parse(request.responseText);
        this.toHtml(joke);
      }
    };
    // verstuur het verzoek naar de server
    request.send();
  }
  toHtml(joke) {
    document.getElementById("category").innerHTML = `Category = ${joke.type}`;
    document.getElementById("setup").innerHTML = joke.setup;
    document.getElementById("punchline").innerHTML = joke.punchline;
  }
}
```

Voorbeeld - JSON

- Resultaat



- Je kan ook bij deze API een categorie meegeven – voorbeeld voor programming:
<https://official-joke-api.appspot.com/jokes/programming/random>
- Probeer dit eventueel uit in het voorbeeld. Let op het result is een array van object(en).

Ajax

- Een Ajax request wordt asynchroon afgehandeld. Hiervoor hebben we twee mogelijkheden: callback functie en promises.
- Tot nu toe hebben we ons Ajax request afgehandeld met een callback functie.
- Dit is niet echt de ideale oplossing.
- We zullen in het volgende deel gebruik maken van promises.

08: Ajax – fetch API

Ajax en promises

Ajax en promises

- In plaats van het verwerken van het Ajax request (opvragen data) te koppelen aan een callback functie, maken we gebruik van een promise.
- We herwerken het vorige voorbeeld (enkel JSON) maar gebruiken nu een promise.
- In plaats van een callback function te koppelen aan het readystatechange event, maken we een promise aan.
- De promise voert het Ajax request uit.
- Afhankelijk van het resultaat van het request krijgen we een resolve (promise succesvol afgehandeld) of een reject (niet succesvol).

Voorbeeld

- Open voorbeeld AjaxPromise

```
function ajaxRequest(url){  
    return new Promise((resolve, reject) => {  
        const request = new XMLHttpRequest();  
        request.open('GET',url );  
        request.onreadystatechange = () => {  
            if (request.readyState === 4){  
                if (request.status === 200) {  
                    resolve(JSON.parse(request.responseText));  
                }  
                else {  
                    reject(`ERROR ${request.status} while processing.`);  
                }  
            }  
        }  
        request.send();  
    });  
}
```

Voorbeeld

- Open voorbeeld AjaxPromise

```
class JokeApp{
  getData(){
    ajaxRequest('https://official-joke-api.appspot.com/random_joke')
      .then(resolveValue => { this.toHtml(resolveValue); })
      .catch(rejectValue => { console.log(rejectValue); });
  }
  toHtml(joke) {
    document.getElementById("category").innerHTML = `Category = ${joke.type}`;
    document.getElementById("setup").innerHTML = `Q: ${joke.setup}`;
    document.getElementById("punchline").innerHTML = `A: ${joke.punchline}`;
  }
}
```

08: Ajax – fetch API

Fetch API

Fetch API

Om een Ajax request uit te voeren moet er steeds dezelfde code gebruikt worden:

- XHR object aanmaken.
- XHR object configureren met open method:
 - http request methode (GET, POST, PUT, DELETE)
 - url
 -
- onreadystatechange event listener:
 - koppelt een callback functie aan dit event.
 - deze functie zal uitgevoerd worden van zodra de status van het request verandert.
- XHR object versturen naar server met send method:
 - verstuurt het request naar de server.
 - data kunnen als parameter naar de server gestuurd worden.

Fetch API

Om dit op een meer gestructureerde manier en compacter te doen is de fetch API ontwikkeld:

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

In deze cursus gaan we dit niet in detail behandelen, maar enkel het gedeelte om een GET Ajax Request te doen.

Fetch API

The Fetch API is vergelijkbaar met de werking van XHR.

- De **fetch()** methode heeft één verplicht argument, het 'path' (vaak url) naar de 'resource' die je wil ophalen.
- Deze methode retourneert een Promise die het antwoord zal verwerken van het 'request', of dit succesvol is of niet.
- Eenmaal het antwoord is ontvangen, zijn er een aantal mogelijkheden om het antwoord (body) te verwerken:
 - `Body.json()`: retourneert een Promise dat de body parsed naar JSON
 - `Body.text()`: retourneert een Promise dat de body parsed naar text (UTF-8)

Voorbeeld

- We hernemen het vorige voorbeeld en passen dit aan. We gebruiken fetch ipv XHR. (Open voorbeeld fetch.js)
- Dit geeft volgende code:

```
function fetchRequest(url){
    return fetch(url)
        .then(body => body.json());
}

class JokeApp{
    getData(){
        fetchRequest('https://official-joke-api.appspot.com/random_joke') //retourneert
        .then(data => this.toHtml(data)) //resolve promise (argument is response)
        .catch(error => console.error(error)); //reject promise (argument is error)
    }
    toHtml(joke) {
        document.getElementById("category").innerHTML = `Category = ${joke.type}`;
        document.getElementById("setup").innerHTML = `Q: ${joke.setup}`;
        document.getElementById("punchline").innerHTML = `A: ${joke.punchline}`;
    }
}
```

08: Ajax – fetch API












Uitgebreid voorbeeld: Countries of the World

Eindresultaat

Alle landen

Countries - capital - region - flag

Number of countries: 250

Country	Capital	Region	Flag
Afghanistan - افغانستان	Kabul	Asia	
Åland Islands - Åland	Mariehamn	Europe	
Albania - Shqipëria	Tirana	Europe	
Algeria - الجزائر	Algiers	Africa	
American Samoa - American Samoa	Pago Pago	Oceania	
Andorra - Andorra	Andorra la Vella	Europe	
Angola - Angola	Luanda	Africa	
Anguilla - Anguilla	The Valley	Americas	
Antarctica - Antarctica		Polar	
Antigua and Barbuda - Antigua and Barbuda	Saint John's	Americas	
Argentina - Argentina	Buenos Aires	Americas	

HO
GENT

Eindresultaat






- We gaan alle landen ophalen bij de API:
<https://restcountries.eu/rest/v2/all>
- We gaan deze objecten mappen naar Country objecten (countryName, capital, region, flag – de countryName bevat de name en de nativeName uit de objecten van de API).
- We geven deze data weer in tabelvorm.
- We voorzien ook een filter via een input veld.

Eindresultaat

Alle landen die beginnen met 'be'

Countries - capital - region - flag

Number of countries: 5

Country	Capital	Region	Flag
Belarus - Беларусь	Minsk	Europe	
Belgium - België	Brussels	Europe	
Belize - Belize	Belmopan	Americas	
Benin - Bénin	Porto-Novo	Africa	
Bermuda - Bermuda	Hamilton	Americas	

HTML – CSS (bootstrap)

```
<body>
  <div class="container">
    <div class="page-header">
      <h1>Countries - capital - region - flag</h1>
    </div>
    <div class="form-group">
      <input type="text" class="form-control" id="search" />
    </div>
    <div class="form-group">
      <h4 id="number"></h4>
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Country</th>
            <th>Capital</th>
            <th>Region</th>
            <th>Flag</th>
          </tr>
        </thead>
        <tbody id="countries">
        </tbody>
      </table>
    </div>
  </div>
</div>
```

JavaScript – Country class

- Constructor met 4 parameters – de members van de klasse.
- De getters van alle members.

```
class Country {  
  constructor(countryName, capital, region, flag) {  
    this._countryName = countryName;  
    this._capital = capital;  
    this._region = region;  
    this._flag = flag;  
  }  
  
  get countryName() {  
    return this._countryName;  
  }  
  
  get capital() {  
    return this._capital;  
  }  
  
  get region() {  
    return this._region;  
  }  
  
  get flag() {  
    return this._flag;  
  }  
}
```

JavaScript – CountriesRepository class

- this._countries: Array die alle country objecten bijhoudt, en getter.
- getCountries(searchstring). filtert de landen adhv de zoekString)

```
class CountriesRepository {  
  constructor(countries) {  
    this._countries = countries;  
  }  
  
  get countries() { return this._countries; }  
  
  getCountries(searchString){  
    return searchString === '' ? this.countries  
      : this.countries.filter((c)=>c.countryName.toLowerCase()  
        .startsWith(searchString.toLowerCase()));  
  }  
}
```

CountryApp

- Zorgt voor het ophalen van de data bij de API (Ajax-Promise).
- Zorgt voor het weergeven van de data op de webpagina (DOM).

```
class CountriesApp {  
  constructor() {  
    this.getData();  
  }  
  getData(){ ...  
  }  
  showCountries = (event) => { ...  
  }  
}
```

Initialisatie van de app: init

- App object aanmaken en bij ingave in de filter (search) het resultaat weergeven van de zoekFilter (input field).

```
const init = function(){  
  const app = new CountriesApp();  
  document.getElementById('search').addEventListener('keyup', app.showCountries);  
}  
  
window.onload = init;
```


CountriesApp: getData() - promiseAjax

```
function ajaxRequest(url){
    return new Promise((resolve, reject) => {
        const request = new XMLHttpRequest();
        request.open('GET', url);
        request.onreadystatechange = () => {
            if (request.readyState === 4){
                if (request.status === 200) {
                    resolve(JSON.parse(request.responseText));
                }
                else {
                    reject(`ERROR ${request.status} while processing.`);
                }
            }
        }
        request.send();
    })
}
```

```
getData(){
    ajaxRequest('https://restcountries.eu/rest/v2/all')
    .then(resultValue => {
        this._countriesRepository =
            new CountriesRepository(resultValue.map(
                r=>new Country(`${r.name} - ${r.nativeName}`, r.capital, r.region, r.flag))
            );
        this.showCountries();
    })
    .catch(rejectValue => { console.log(rejectValue); });
}
```

- Gebruik maken van een promise om het AJAX-request af te werken.
- Bij succes (resolve) worden de data van de API gemapt naar Country Objecten (en als parameter aan de repository constructor meegegeven) en vervolgens worden de countries weergegeven op de pagina.
- Indien niet succesvol, wordt een melding naar de console weggeschreven.

CountriesApp: getData() - fetch

```
function fetchRequest(url){  
    return fetch(url)  
        .then(body => body.json());  
}
```

```
getData(){  
    ajaxRequest('https://restcountries.eu/rest/v2/all')  
        .then(resultValue => {  
            this._countriesRepository =  
                new CountriesRepository(resultValue.map(  
                    r=>new Country(`${r.name} - ${r.nativeName}`, r.capital, r.region, r.flag))  
                );  
            this.showCountries();  
        })  
        .catch(rejectValue => { console.log(rejectValue); });  
}
```

- Gebruik maken van een promise om het AJAX-request af te werken.
- Bij succes (resolve) worden de data van de API gemapt naar Country Objecten (en als parameter aan de repository constructor meegegeven) en vervolgens worden de countries weergegeven op de pagina.
- Indien niet succesvol wordt een melding naar de console weggeschreven.

CountriesApp: showCountries()

```
showCountries = (event) => {  
  const val = typeof(event) === 'undefined'?':': event.target.value;  
  const countries = this.countriesRepository.getCountries(val);  
  document.getElementById("countries").innerHTML = '';  
  document.getElementById("number").innerText= `Number of countries: ${countries.length}`;  
  countries.forEach((c) => {  
    const tr = document.createElement("tr");  
    const td1 = document.createElement("td");  
    td1.appendChild(document.createTextNode(c.countryName));  
    const td2 = document.createElement("td");  
    td2.appendChild(document.createTextNode(c.capital));  
    const td3 = document.createElement("td");  
    td3.appendChild(document.createTextNode(c.region));  
    const td4 = document.createElement("td");  
    const icon = document.createElement("img");  
    icon.src = c.flag;  
    icon.width = "35";  
    icon.height = "25";  
    td4.appendChild(icon);  
    tr.appendChild(td1);  
    tr.appendChild(td2);  
    tr.appendChild(td3);  
    tr.appendChild(td4);  
    document.getElementById("countries").appendChild(tr);  
  });  
}
```

- Via DOM wordt dynamisch de tabel waarin de countries worden weergegeven opgebouwd.
- De eerste regel is nodig voor het gebruik van de tekstZoek filter.