HoGent

BEDRIJF
EN
ORGANISATIE

Ajax en Promises

Asynchronous Javascript And XML

Ajax

Synchroon vs Asynchroon

Wat is Ajax?

Waarom Ajax?

Werking van Ajax.

Voorbeelden.

- Response tekst
- Response json

HoGent

HoGent

Synchroon vs Asynchroon

- In Javascript hebben krijgen we vaak te maken met asynchroon gedrag.
- Als je in synchrone programma's twee regels code hebt (L1 gevolgd door L2), dan kan L2 pas gestart worden als L1 klaar is met uitvoeren.
- Je kunt je dit voorstellen alsof je in een rij mensen staat te wachten om een treinkaartje te kopen. Je kunt pas beginnen met het kopen van een treinkaartje voordat alle mensen voor jou klaar zijn met kopen. Evenzo kunnen de mensen achter je niet beginnen met het kopen van hun kaartjes totdat jij het jouwe hebt gekocht.

Synchroon vs Asynchroon

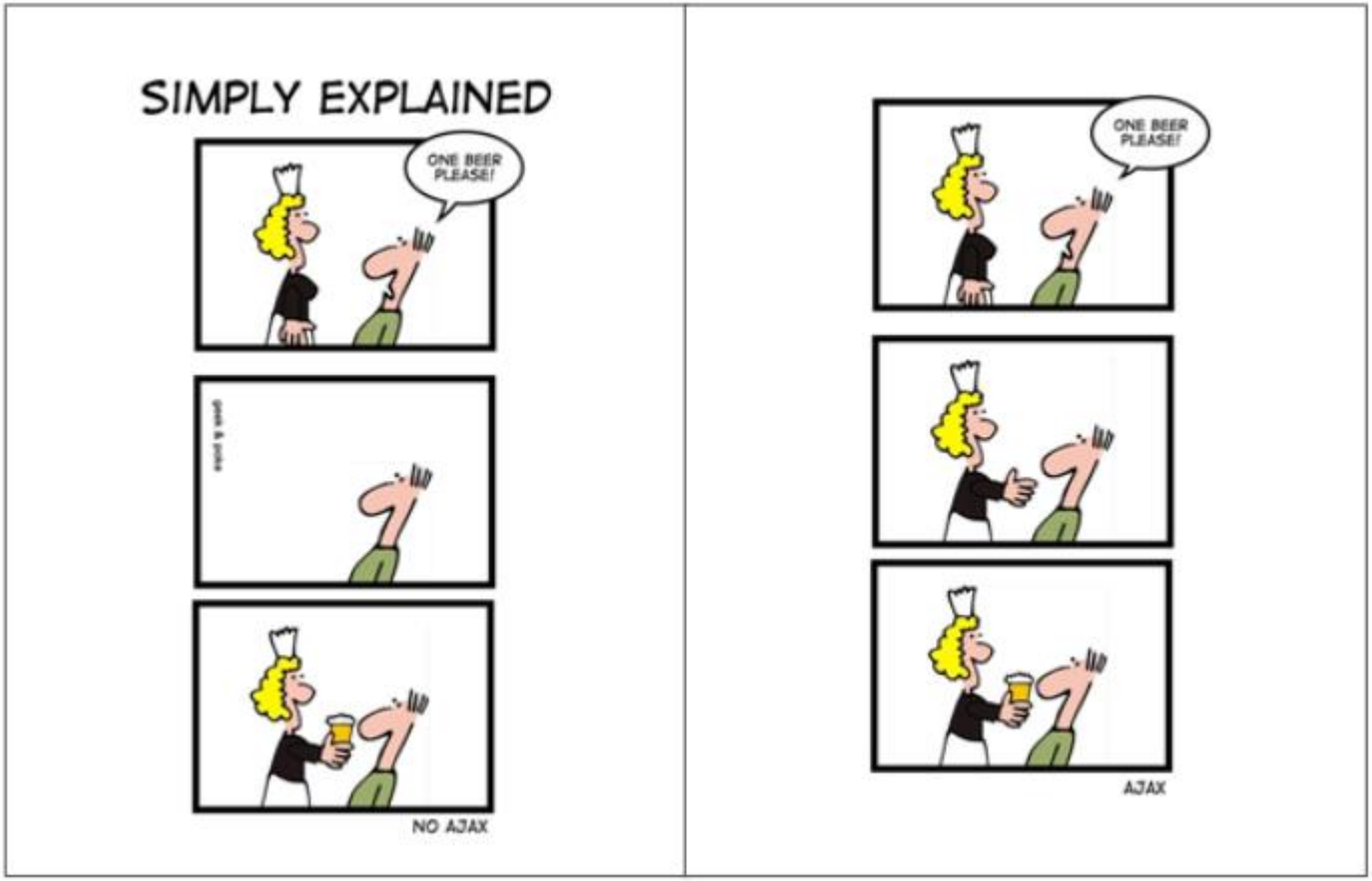
- In asynchrone programma's kan je twee regels code hebben (L1 gevolgd door L2), waarbij L1 een taak plant die in de toekomst moet uitgevoerd worden, maar L2 zal uitgevoerd zijn voordat die taak voltooid is.
- Je kunt je dit voorstellen als dat je in een restaurant aan het eten bent. Andere mensen bestellen hun eten. Jij kunt ook je eten bestellen. Je hoeft niet te wachten tot ze hun eten hebben gekregen en het hebben opgegeten voordat je bestelt. Evenzo hoeven andere mensen niet op jou te wachten tot jij je eten hebt gekregen en opgegeten voordat ze kunnen bestellen. Iedereen krijgt zijn eten zodra het klaar is.

Synchroon vs Asynchroon

- De volgorde waarin mensen hun eten krijgen, hangt vaak samen met de volgorde waarin ze eten hebben besteld, maar deze sequenties hoeven niet altijd identiek te zijn. Als jij bijvoorbeeld een biefstuk bestelt en ik daarna een glas water bestel, ontvang ik waarschijnlijk eerst mijn bestelling, omdat het meestal niet zo veel tijd kost om een glas water te serveren als om een biefstuk klaar te maken.

Synchroon vs Asynchroon

- The Geek & Poke explanation voor Synchroon vs Asynchroon



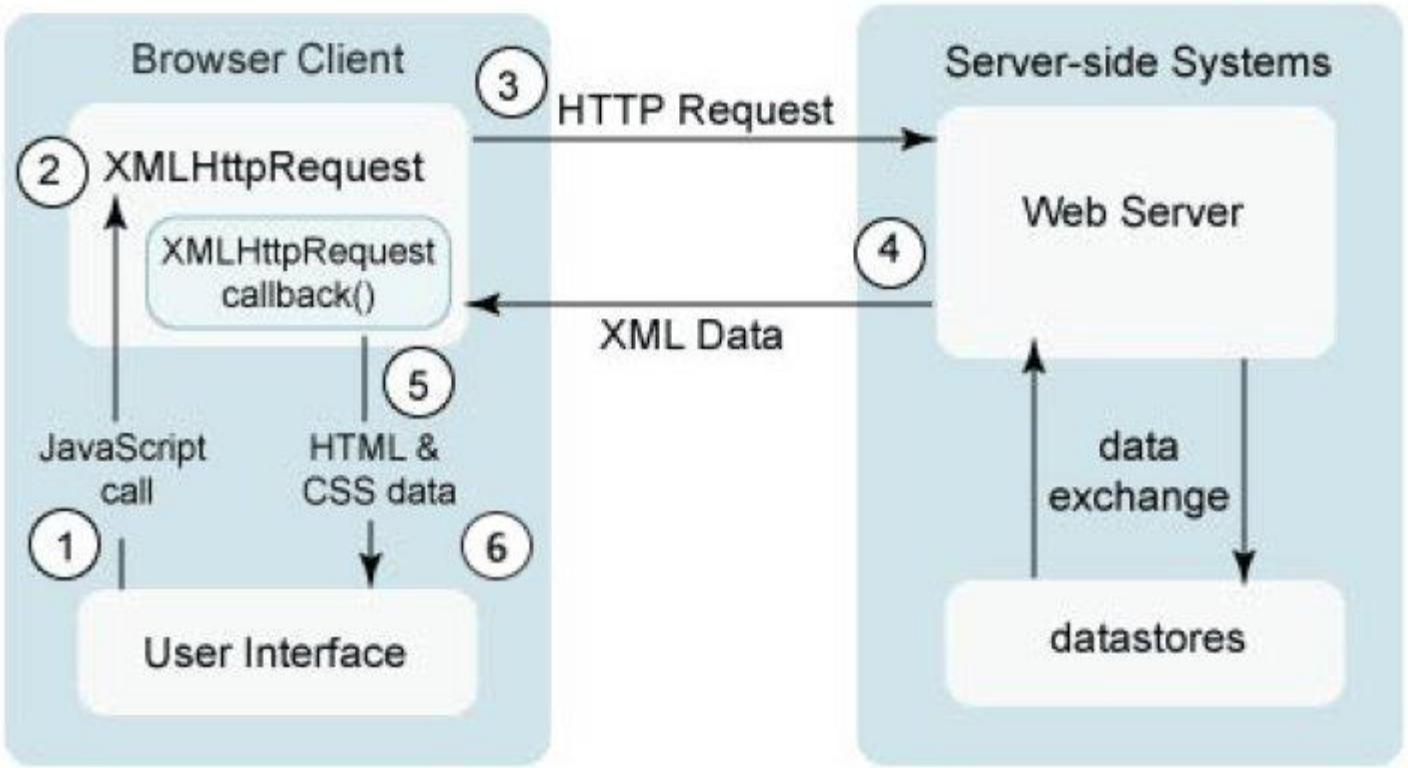
AJAX (spreek uit: eɪdʒæks)

- **Ajax** is een verzameling van technologieën, die gebruikt worden aan de client-side om asynchrone web applicaties te ontwikkelen:
  - 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, XSLT en JSON (JavaScript Object Notation)voor de opslag, aanpassing en transport van gegevens.
  - Het XMLHttpRequest object voor asynchrone communicatie met de back-end server.
  - JavaScript om alles aan elkaar te binden.

AJAX (spreek uit: eɪdʒæks)

- 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.
- 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 uitgevonden 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


Browser Client
3
HTTP Request
Server-side Systems
2
XMLHttpRequest
XMLHttpRequest
callback()
Web Server
4
XML Data
5
JavaScript
call
HTML &
CSS data
data
exchange
1
6
User Interface
datastores

Waarom 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: http://www.google.com/webhp?complete=1&hl=nl

Werking AJAX

- Voordelen:
  - 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.
  - De gebruiker kan verder werken terwijl de pagina wordt aangepast.

Ajax – eenvoudig voorbeeld

- Om Ajax requests te illustreren hebben we een backend nodig die data kan teruggeven
- Om zelf geen backend te moeten schrijven gaan we een joke API (https://08ad1pao69.execute-api.us-east-1.amazonaws.com/dev/random_joke) gebruiken. Er zijn er nog andere.
- Deze API retourneert een grap maar is -vooral- volledig vrij te gebruiken zonder dat we ook authenticatie nodig hebben
- Andere interessante API's zijn te vinden op https://github.com/toddmotto/public-apis

Ajax – een eenvoudig voorbeeld- tekst

- Open AjaxPromisesVoorbeelden > voorbeeld1
- De code van index.html ziet er als volgt uit

```
<div id="wrapper">
    <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>
```

Ajax – eenvoudig voorbeeld - tekst

```
function ajax_get() {
  // creatie van het XHR object
  const request = new XMLHttpRequest();
  // open(Methode, URL, Asynchronous, Username, Password)
  request.open('GET',
'https://08ad1pao69.execute-api.us-east1.amazonaws.com/dev/random_joke');
  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();
}

function init() {
  document.getElementById("joke").onclick = () => {ajax_get();}
}

window.onload = () => init();
```

{"id":7,"type":"general","setup":"What do you call a singing Laptop","punchline":"A Dell"}

Ajax – XHR object - methodes

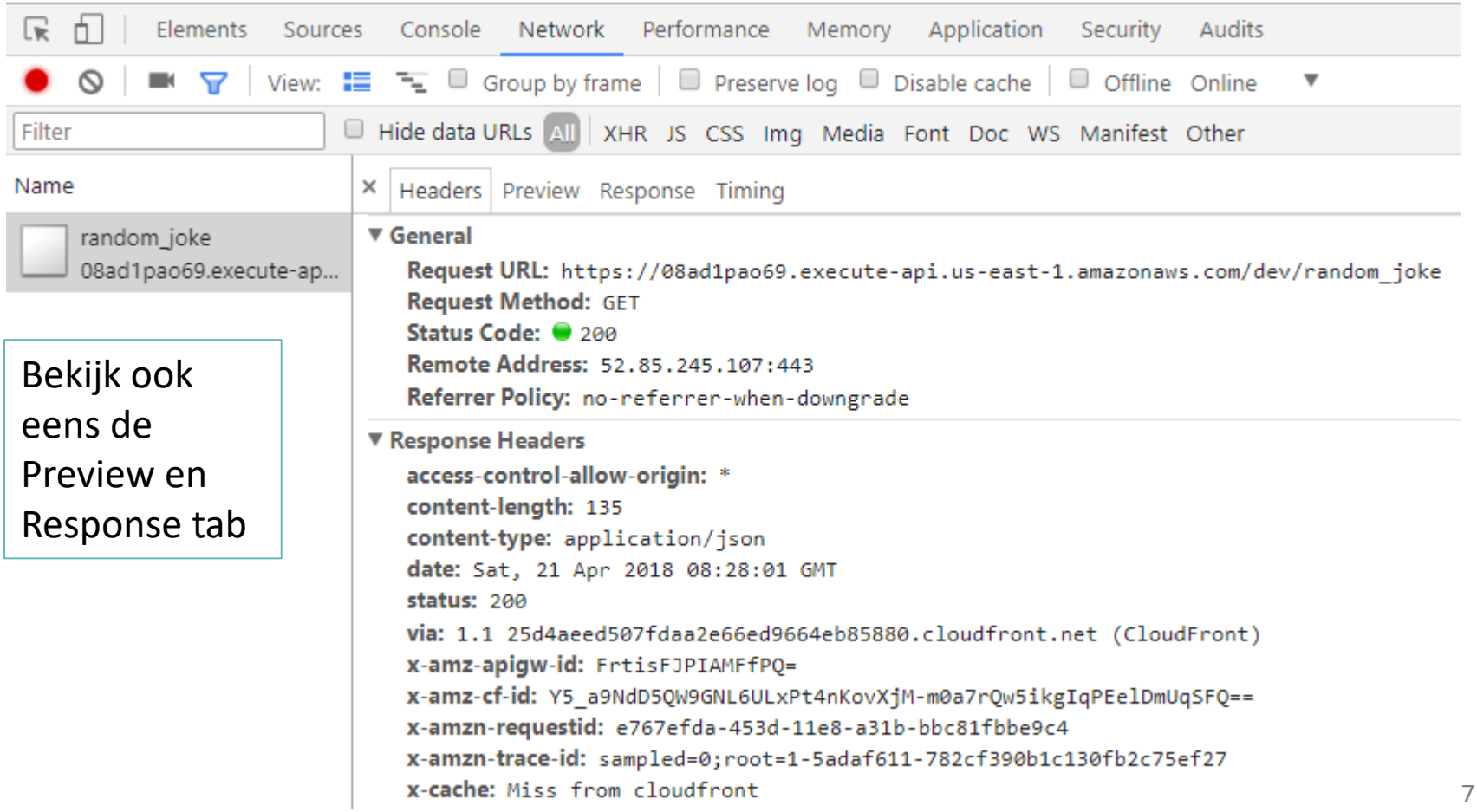
- open method: configureert het XHR object.
  open(Method, URL, Asynchronous, UserName, Password)
  - HTTP request method: GET – POST – HEADER - ….
  - URL: de url waar het request naar verstuurd wordt (het adres). Dit kan een tekst, json, xml bestand zijn. Het kan ook een server-side script zijn (C# - 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.
- onreadystatechange event listener: koppelt een callback functie aan dit event. Deze functie zal uitgevoerd worden van zodra de server op het request geantwoord heeft.
- 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:
    - 0: request not initialized
    - 1: server connection established
    - 2: request received by server
    - 3: processing request by server
    - 4: request finished and response is ready
  - status: geeft de http status die door de server is teruggegeven:
    - 200: "OK“
    - 404: Page not found
  - 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.
  - responseXML: de server geeft de data terug als xml.

Ajax – een eenvoudig voorbeeld

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



Bekijk ook eens de Preview en Response tab

Ajax – een eenvoudig voorbeeld

- Als je via Chrome Developer Tools de Preview tab bekijkt, zie je dat er een JSON object geretourneerd wordt. Daarom zullen we de code aanpassen: zie volgende slide.
- De code van index.html ziet er als volgt uit

```
<div id="wrapper">
    <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>
```

Ajax – eenvoudig voorbeeld - JSON

```
function ajax_get() {
  // creatie van het XHR object
  const request = new XMLHttpRequest();
  // open(Methode, URL, Asynchronous, Username, Password)
  request.open('GET',
         'https://08ad1pao69.execute-api.us-east-1.amazonaws.com/dev/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) {
      // Er wordt een object literal gemaakt van het antwoord
      const joke = JSON.parse(request.responseText);
      toHtml(joke);
    }
  };
  // verstuur het verzoek naar de server
  request.send();
}

function toHtml(joke) {
  document.getElementById("category").innerHTML = `Category = ${joke.type}`;
  document.getElementById("setup").innerHTML = joke.setup;
  document.getElementById("punchline").innerHTML = joke.punchline;
}

function init() {
  document.getElementById("joke").onclick = () => {ajax_get();}
}

window.onload = () => init();
```

JSON.parse hebben we ook gebruikt bij Webstorage!!

Promises

HoGent

Promises

- Het idee achter promises is al oud (jaren 70), maar men is ze pas echt beginnen gebruiken de laatste ~ 5 jaar
- De meeste programmeertalen kennen het concept ondertussen, hoewel ze vaak 'futures' genoemd worden (C++, Java, ...)
- Het idee is dat je het resultaat van een asynchrone operatie door een variabele laat voorstellen, dat je dan kan doorgeven en waarmee je op elk moment kan interageren
- Een promise is een placeholder voor het resultaat van een asynchrone operatie

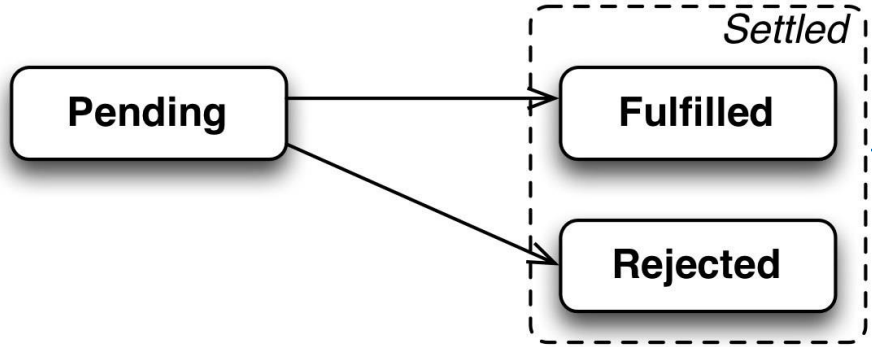
Promises

- Stel dat je een topzanger bent en fans vragen je dag en nacht wanneer je volgende single verschijnt.
- Om niet meer constant lastig gevallen te worden, geef je een lijst aan je fans waarop ze hun gegevens kunnen achterlaten en wanneer je volgende single verschijnt, zullen ze gecontacteerd worden.
- Ook wanneer er iets zou foutlopen en er komt geen single meer, kunnen ze op die manier op de hoogte gebracht worden

Promises

- Dit is een voorbeeld uit het echte leven voor iets wat zich vaak voordoet in programmeren.
- Er is de “producing code” die iets doet en daarvoor tijd nodig heeft, bijvoorbeeld data ophalen op een server. In ons voorbeeld is dat de zanger.
- Er is “consuming code” die de resultaten wil ontvangen wanneer die beschikbaar zijn. In ons voorbeeld zijn dat de fans.
- Een promise is een speciaal JavaScript object dat beide linkt. In ons voorbeeld is dat de lijst.

Promises



- Promises kunnen zich in 3 toestanden bevinden:
  - Pending: Het resultaat van de promise is nog niet bepaald aangezien de asynchrone operatie die verantwoordelijk is voor het resultaat, nog niet afgerond is.
  - Fulfilled: De asynchrone operatie is afgerond en de promise heeft een waarde.
  - Rejected: De asynchrone operatie is niet goed afgerond waardoor de promise niet kan gerealiseerd worden. In deze status heeft de promise een reden die aangeeft waarom de operatie faalde.
- Wanneer een promise ‘pending’ is, kan het overgaan naar de toestand ‘fulfilled’ of ‘rejected’. Eens een promise ‘fulfilled’ of ‘rejected’ is, kan het niet meer overgaan naar een andere status. Zijn value of de reden waarom de operatie faalde, zal niet meer veranderen.

Promises

- De algemene structuur van een Promise

```
const p = new Promise((resolve, reject) => {

    // Voer een asynchrone taak uit en vervolgens ...

    if(/* De asynchrone taak is gelukt */) {
        resolve(resolveWaarde);
    }
    else {
        reject(rejectWaarde);
    }
});

p.then(() => {
    /* Doe iets met het resultaat.
    Hier komt de actie die moet uitgevoerd worden als het resultaat
    beschikbaar is. */
}).catch(() => {
    /* Error :(
        Hier komt de actie die moet uitgevoerd als de taak faalde */
})
```

p dus geen gewone variabele die een waarde bevat, maar een object waarop je functies kan aanroepen om iets te doen met een toekomstig resultaat, namelijk then en catch

Promises - Voorbeeld

- We willen een eenvoudig spelletje implementeren met behulp van promises
  - Als iemand een correcte gok doet voor een cijfer tussen 1 en 3, wint hij een haardroger.
  - Als iemand vervolgens een correcte gok doet voor een cijfer tussen 1 en 5, wint hij bovendien een koelkast.
  - Als hij vervolgens opnieuw een correcte gok doet voor een cijfer tussen 1 en 8, wint hij ook nog eens een vliegreis.
- Initieel zullen we een nieuwe promise creëren die de gebruiker een gokje laat wagen (via prompt) en vervolgens een random getal bepaalt.
- Als de gebruiker correct gokt, wordt de promise fulfilled. Anders wordt de promise rejected.

Promises – Voorbeeld – Stap 1

- Open AjaxPromisesVoorbeelden > voorbeeld2

```
<div id="wrapper">
    <div class="page-header">
        <h1>Doe een gok</h1>
    </div>

    <div class="form-group">
        <p>
            <input type="button" id="gok" value="Doe een gok"
                   class="btn btn-default">
        </p>
    </div>
</div>
```

Promises – Voorbeeld – Stap 1

```
const promiseGok1 = new Promise((resolve, reject) => {

    const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
    const randomWaarde = Math.floor(Math.random() * 3) + 1;

    if (gok === randomWaarde) {
        resolve(randomWaarde);
    }
    else {
        reject(randomWaarde);
    }
})

promiseGok1.then((resolveWaarde) => {
    console.log("Juist gegokt! U hebt een haardroger gewonnen!")
});
promiseGok1.catch((rejectWaarde) => {
    console.log(`Fout gegokt! De waarde op de dobbelsteen was ${rejectWaarde}`);
});
```

Promises – Voorbeeld – Stap 1

- Je kan deze code ook schrijven als

```
const promiseGok1 = new Promise((resolve, reject) => {

    const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
    const randomWaarde = Math.floor(Math.random() * 3) + 1;

    if (gok === randomWaarde) {
        resolve(randomWaarde);
    }
    else {
        reject(randomWaarde);
    }
}).then((resolveWaarde) => {
    console.log("Juist gegokt! U hebt een haardroger gewonnen!")
}).catch((rejectWaarde) => {
    console.log(`Fout gegokt! De waarde op de dobbelsteen was ${rejectWaarde}`);
});
```

Promises – Voorbeeld – Stap 1

- De promise wordt DIRECT gecreëerd, maar de promise weet nog niet wat het resultaat gaat zijn
- Je stelt in wat er moet gebeuren bij het (toekomstig) resultaat, door een callback functie mee te geven aan de then method
- Deze zal aangeroepen worden op het moment dat de asynchrone functie “correct” afgewerkt is.
- De then callback doorgeven doet op zich niets, en is niet blocking.
- De code die moet uitgevoerd worden op het moment dat de asynchrone code “niet correct” afgewerkt is, doe je door ook een catch te definiëren

Promises – Voorbeeld – Stap 1

- Op het eerste zicht lijkt dit erg op callbacks gebruiken: je geeft een functie mee die zal aangeroepen worden als een asynchrone functie gedaan heeft met werken
- Het verschil is subtiel, maar belangrijk. Door een 'extra level of indirection' toe te voegen kan je je code veel beter structuren
- Én omdat je een promise variabele hebt voor elke stap in een aaneenschakeling van asynchrone functies, kan je veel makkelijker op elk moment verschillende wegen uitgaan

Promises – Voorbeeld – Stap 1

- Promises zijn intern eigenlijk kleine state machines, die een zekere levensloop hebben
- Als ze aangemaakt worden, starten ze altijd in de pending state, de asynchrone code is aan het lopen, de then en catch functies worden niet gestart maar onthouden
- Eens de asynchrone code afgewerkt is, komt de promise ofwel in de fulfilled state of in de rejected state, alnaargelang de resolve of reject aangeroepen was
- En dan zullen alle then of catch functies één voor één aangeroepen worden

Promises – Voorbeeld – Stap 2

- De voorgaande code wordt onmiddellijk uitgevoerd, als de pagina geladen wordt, dus nog voor we op de knop Doe een gok hebben geklikt.
- Om dit op te lossen, plaatsen we het creëren van de promise in een functie.
- Als op de knop Doe een gok geklikt wordt, wordt deze functie uitgevoerd => pas dan wordt de promise gecreëerd

Promises – Voorbeeld – Stap 2

```
const gok1 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
        const randomWaarde = Math.floor(Math.random() * 3) + 1;

        if (gok === randomWaarde) {
            resolve(randomWaarde);
        }
        else {
            reject(randomWaarde);
        }
    }).then(() => {
        console.log("Juist gegokt! U hebt een haardroger gewonnen!");
    }).catch((rejectWaarde) => {console.log(`Fout gegokt!
                De waarde op de dobbelsteen was ${rejectWaarde}`) })
}

function init() {
    document.getElementById("gok").onclick = () => {
        gok1();
    }
}

window.onload = () => init();
```

Promises – Voorbeeld – Stap 3

- We breiden het voorbeeld uit.
- Nadat de gebruiker de eerste keer juist gegokt heeft, wint hij niet enkel een haardroger, maar moet hij de kans krijgen te gokken voor de koelkast.
- Als hij ook voor de koelkast correct gegokt heeft, moet hij de kans krijgen te gokken voor de vliegreis.
- Daarom roepen we binnen .then respectievelijk de functie gok2() en gok3() op. Deze functies retourneren een promise die toelaat te gokken voor de koelkast, respectievelijk de vliegreis.

```
Juist gegokt! U hebt een haardroger gewonnen!
Juist gegokt! U hebt ook een koelkast gewonnen!
Juist gegokt! U hebt ook een vliegreis gewonnen!
```

Promises – Voorbeeld – Stap 3

```
const gok1 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
        const randomWaarde = Math.floor(Math.random() * 3) + 1;

        if (gok === randomWaarde) {
            resolve(randomWaarde);
        }
        else {
            reject(randomWaarde);
        }
    }).then(() => {
        console.log("Juist gegokt! U hebt een haardroger gewonnen!");
        gok2(); // Hierdoor wordt de promise van gok2 geretourneerd
    }).catch(() => {console.log("Fout gegokt!
                U hebt de haardroger niet gewonnen!")
 })
}
```

Promises – Voorbeeld – Stap 3

```
const gok2 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 5"));
        const randomWaarde = Math.floor(Math.random() * 5) + 1;

        if (gok === randomWaarde) {
            resolve(randomWaarde);
        }
        else {
            reject(randomWaarde);
        }
    }).then(() => {
        console.log("Juist gegokt! U hebt ook een koelkast gewonnen!");
        gok3(); // Hierdoor wordt de promise van gok3 geretourneerd
    }).catch(() => {console.log("Fout gegokt!
                U hebt de koelkast niet gewonnen!")
 })
}
```

Promises – Voorbeeld – Stap 3

```
const gok3 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 8"));
        const randomWaarde = Math.floor(Math.random() * 8) + 1;

        if (gok === randomWaarde) {
            resolve(randomWaarde);
        }
        else {
            reject(randomWaarde);
        }
    }).then(() => {
        console.log("Juist gegokt! U hebt ook een vliegreis gewonnen!");
    }).catch(() => {console.log("Fout gegokt!
                U hebt de vliegreis niet gewonnen!")
 })
}

function init() {
    document.getElementById("gok").onclick = () => {gok1();}
}

window.onload = () => init();
```

Promises – Voorbeeld – Stap 3

- In het voorbeeld wordt binnen de afhandeling van de promise een andere promise geretourneerd.
- In dat – bijzondere – geval krijgt de oorspronkelijke promise nu dezelfde state als diegene die geretourneerd wordt, met dezelfde value
- Als binnen de afhandeling echter een gewone waarde wordt geretourneerd, komt de promise in fulfilled state of in de rejected state, alnaargelang de resolve of reject aangeroepen was. De waarde is de resolve waarde, respectievelijk de reject waarde.

Promise.all

- Een van de problemen met callbacks was een functie pas uitvoeren nadat verschillende andere (parallelle) callbacks uitgevoerd waren
- Een gelijkaardig probleem is een functie pas uitvoeren nadat de eerste van een reeks parallelle callbacks uitgevoerd is
- Voor beide problemen heeft promise een oplossing

Promise.all

- Het probleem is als volgt, we hebben een aantal (ongerelateerde) promises die we tesamen willen starten, en we willen een andere functie uitvoeren als ze ALLEMAAL afgerond zijn
- Promise.all is een statische functie die net dat bereikt: je geeft er een array van promises aan mee, en hij wordt geresolved met een array van values als ze allemaal afgerond zijn. Je krijgt dus een array met alle values
- Als er een faalt, dan faalt de volledige Promise.all (en hij faalt al zodra de eerste faalt 'fail fast')
- Open AjaxPromisesVoorbeelden > voorbeeld3

Promise.all

```
const gok1 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
        const randomWaarde = Math.floor(Math.random() * 3) + 1;
        gok === randomWaarde ? resolve("Haardroger") : reject("Geen haardroger");
    })
}

const gok2 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 5"));
        const randomWaarde = Math.floor(Math.random() * 5) + 1;
        gok === randomWaarde ? resolve("Koelkast") : reject("Geen koelkast");
    })
}

const gok3 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 8"));
        const randomWaarde = Math.floor(Math.random() * 8) + 1;
        gok === randomWaarde ? resolve("Vliegreis") : reject("Geen vliegreis");
    })
}
```

Promise.all

```
function init() {
    document.getElementById("gok").onclick = () => {
        Promise.all([gok1(), gok2(), gok3()]).then(values => {
            console.log(values);
        }).catch(values => {console.log(values)});
    }
}

window.onload = () => init();
```

```
// Als enkel gok2 fout is => catch
Geen koelkast

// Als geen enkele gok fout is => then
▶ (3) ["Haardroger", "Koelkast", "Vliegreis"]

// Als enkel gok3 fout is => catch + fail fast
Geen vliegreis

// Als gok1, gok2 en gok3 fout zijn => catch + fail fast
Geen haardroger
```

Promise.race

- Een gelijkaardige functie aan Promise.all is Promise.race
- Als één van de promises in de array resolved is, dan is de race afgelopen en resolved de promise race ook
- Maar ook als er één faalt, faalt de Promise.race (dus het gaat niet verder tot er één resolved)
- Kort samengevat, als er één afgerond is, is de promise.race ook afgerond in dezelfde state met dezelfde waarde

Promise.race

```
const gok1 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 3"));
        const randomWaarde = Math.floor(Math.random() * 3) + 1;
        gok === randomWaarde ? resolve("Haardroger") : reject("Geen haardroger");
    })
}

const gok2 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 5"));
        const randomWaarde = Math.floor(Math.random() * 5) + 1;
        gok === randomWaarde ? resolve("Koelkast") : reject("Geen koelkast");
    })
}

const gok3 = () => {
    return new Promise((resolve, reject) => {
        const gok = Number.parseInt(prompt("Geef een getal tussen 1 en 8"));
        const randomWaarde = Math.floor(Math.random() * 8) + 1;
        gok === randomWaarde ? resolve("Vliegreis") : reject("Geen vliegreis");
    })
}
```

Promise.race

```
function init() {
    document.getElementById("gok").onclick = () => {
        Promise.race([gok1(), gok2(), gok3()]).then(values => {
            console.log(values);
        }).catch(values => {console.log(values)});
    }
}

window.onload = () => init();


// Als enkel gok2 fout is => then (owv gok1)
Haardroger

// Als geen enkele gok fout is => then (owv gok1)
Haardroger

// Als enkel gok1 fout is => catch (owv gok1)
Geen haardroger

// Als gok1, gok2 en gok3 fout zijn => catch (owv gok1)
Geen haardroger
```

Promises beperkingen

- Promises maken het werken met asynchrone code een stuk aangenamer, perfect zijn ze natuurlijk niet
- Een eerste probleem is dat een promise altijd start, en altijd afgerond wordt, dus ook als er nooit iemand .then() aanroept, of als het resultaat je niet langer interesseert kan je een promise niet 'cancel'-en
- Eenmaal een promise afgerond is, kan je ze ook niet makkelijk opnieuw uitvoeren, de promise zelf bevat enkel het resultaat
- Om deze problemen aan te pakken hebben we een andere constructie nodig: OBSERVABLES (→ 2TIN)

Ajax en Promises

HoGent

Ajax en Promises

- We gaan het voorbeeld met de grapjes nu herschrijven zodat we gebruik maken van een promise
- Het belangrijkste is dat als je nu een resultaat hebt, dan roep je resolve op met dat resultaat
- De code die moet uitgevoerd worden als er een resultaat is, zet je in .then()
- Als er errors opduiken geef je ze door aan reject

Ajax en Promises

```
function promiseGetRequest(url) {
  // De constructor van een Promise heeft één argument:
  // een functie met 2 parameters: resolve en reject
  return new Promise((resolve, reject) => {
    const request = new XMLHttpRequest();
    request.open('GET', url);
    request.onreadystatechange = () => {
      if (request.readyState === 4) {
        // Als je een resultaat hebt, dan roep je resolve op met dat resultaat
        if (request.status === 200) {
          resolve(JSON.parse(request.responseText));
        }
        // Als er errors opduiken geef je ze door aan reject
        else {
          reject(`ERROR ${request.status} while processing ${url}`);
        }
      }
    };
    request.send();
  }).then(resolveWaarde => { toHtml(resolveWaarde); })
    .catch((rejectWaarde) => { console.log(rejectWaarde); });
}
```

Ajax en Promises

```
function toHtml(joke) {
  document.getElementById("category").innerHTML = `Category = ${joke.type}`;
  document.getElementById("setup").innerHTML = joke.setup;
  document.getElementById("punchline").innerHTML = joke.punchline;
}

function init() {
  document.getElementById("joke").onclick = () => {
    promiseGetRequest(
    'https://08ad1pao69.execute-api.us-east-1.amazonaws.com/dev/random_joke')
  }
}

window.onload = () => init();
```

Ajax en Promises

- Waarom werkt het volgende niet?

```
function promiseGetRequest(url) {
  // De constructor van een Promise heeft één argument:
  // een functie met 2 parameters: resolve en reject
  return new Promise((resolve, reject) => {
    const request = new XMLHttpRequest();
    request.open('GET', url);
    request.onreadystatechange = () => {
      if ((request.readyState === 4) && (request.status === 200)){
        // Als je een resultaat hebt, dan roep je resolve op met dat resultaat
          resolve(JSON.parse(request.responseText));
       }
       // Als er errors opduiken geef je ze door aan reject
      else {
         reject(`ERROR ${request.status} while processing ${url}`);
      }
    };
    request.send();
  }).then(resolveWaarde => { toHtml(resolveWaarde); })
    .catch((rejectWaarde) => { console.log(rejectWaarde); });
}
```

Oefening

HoGent

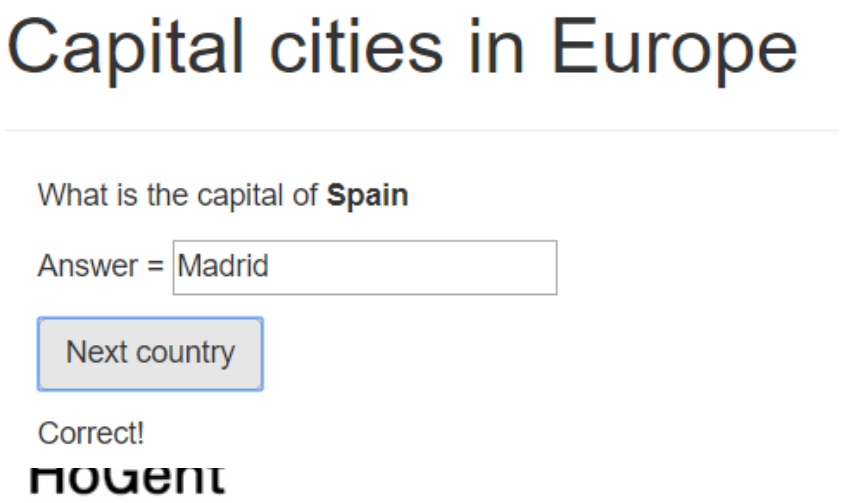
HoGent

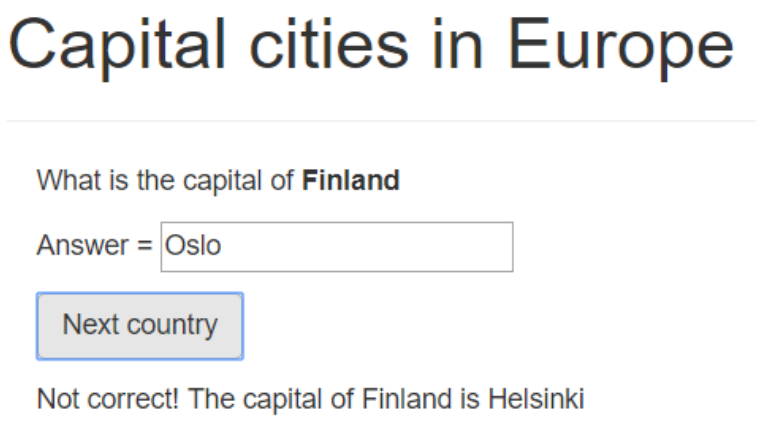
Oefening

- In deze oefening willen we een aantal dingen uit de voorbije lessen samenbrengen
  - Een klasse schrijven
  - Een repository klasse schrijven
  - Het gebruik van een afzonderlijke klasse om data op te halen (met behulp van Ajax en Promises) en te laten zien
- Let op de structuur van deze oefening. De andere oefeningen zullen een analoge structuur hebben.

Oefening

- We willen een applicatie maken waarmee de kennis van de hoofdsteden van Europa opgefrist kan worden
- Om zelf geen backend te moeten schrijven gaan we een countries API gebruiken (https://restcountries.eu/rest/v2/region/europe).
- Bekijk de structuur van het JSON object dat geretourneerd wordt door deze API.





Oefening

- Open AjaxPromisesVoorbeelden > voorbeeld5

```
<div id="wrapper">
        <div class="page-header">
            <h1>Capital cities in Europe</h1>
        </div>
        <div class="form-group">
            <div class="col-sm-5">
                <p>What is the capital of
                    <b>
                        <span id="country"></span>
                    </b>
                </p>
                <p>Answer = <input type="text" id="capital" required></p>
                <p>
                    <input type="button" id="navigation" value="Check"
                           class="btn btn-default">
                </p>
                <p id="feedback"></p>
            </div>
        </div>
</div>
```

Oefening

- Schrijf een klasse CountryInfo die de naam van een land en zijn hoofdstad kan bevatten
- Voorzie getters en setters

Oefening

- Schrijf een klasse CountriesRepository die een array van CountryInfo objecten zal bevatten
- Voorzie getters en setters
- Voorzie een functie addCountry (country, capital)
- Voorzie een functie getRandomCountry die een willekeurig object CountryInfo retourneert
- Om te testen
  - Creëer een object van de klasse CountriesRepository in de init – functie
  - Voeg er een 3 – tal landen met de hoofdsteden aan toe
  - Test de functie getRandomCountry uit. Schrijf weg naar de console.

Oefening

- Schrijf een klasse CountriesComponent die een countriesRepository en een randomCountry heeft
- Voorzie getters en setters
- Voorzie een functie fillCountries(url) die de repository opvult.
- Test dit uit door code toe te voegen aan de init - functie

Oefening

- Voorzie een functie setRandomCountry die een waarde geeft aan het veld randomCountry
- Voorzie een functie showRandomCountry waarbij de naam van het land uitgeschreven wordt in het element met id country
- Pas de code van fillCountries aan zodat deze beide vorige functies op de juiste plaats worden aangeroepen nadat de data geladen is in de repository
- Voorzie een functie checkAnswer waarbij het ingegeven antwoord gecontroleerd wordt (kleine letters of hoofdletters spelen geen rol) en er feedback verschijnt in het element met id feedback

Oefening

- Voeg code toe aan de init – functie
  - Als het opschrift van de knop met id navigation "Check" is
    - Het antwoord wordt gecontroleerd
    - Het opschrift van de knop wordt veranderd naar "Next country"
  - Anders
    - Er wordt een nieuw random land gekozen en getoond
    - De nodige elementen worden leeg gemaakt
    - Het opschrift van de knop wordt veranderd naar Check