



Web scripting

Fetch

**DE HOGESCHOOL
MET HET NETWERK**

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook



Herhaling: JSON

'JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is *easy for humans to read and write*. It is easy for machines to *parse* and *generate*.'

JSON: objects & arrays

object: { "id" : 1, "name" : "tim" }

array: [1, 2, 3]

Combinaties van objects en arrays:

```
[{"id":1, "name":"tim"}, {"id":2, "name":"sofie"}]  
{"id":1, "name":"tim", "hobbies":["tennis", "lezen"]}
```

Herhaling: JSON

JSON.stringify: vorm een object om naar een JSON-string

JSON.parse: ontleed een JSON-string, maak er een object van

```
let person = {  
  name: "tim",  
  hobbies: ["reading", "running", "tennis"]  
};  
let personJSON = JSON.stringify( person );  
console.log( personJSON );  
let person2 = JSON.parse( personJSON );  
console.log(person2);
```

```
{ "name": "tim", "hobbies": ["reading", "running", "tennis"] }  
{ name: 'tim', hobbies: [ 'reading', 'running', 'tennis' ] }
```



Herhaling: arrow notation

```
let sum = ( x, y ) => { return x+y; }
```

```
let minus = ( x, y ) => x-y;
```

```
let inverse = x => 1/x;
```

```
let print = ( text ) => { console.log(text); }
```

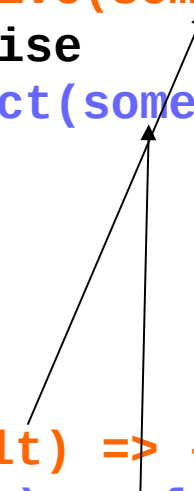
```
print(sum(1,2));  
print(minus(4,5));  
print(inverse(5));
```

Promises

Asynchrone actie:

```
let promise = new Promise( (resolve, reject) => {  
  // do something asynchronous  
  // if success  
  //   resolve(somevalue);  
  // otherwise  
  //   reject(somevalue);  
} );
```

```
promise  
  .then(  
    (result) => {console.log(result); },  
    (error) => {console.log(error); }  
  )
```



The diagram consists of two arrows. The first arrow originates from the `(result)` parameter in the first callback function of the `.then()` method and points to the `resolve(somevalue)` line in the Promise constructor. The second arrow originates from the `(error)` parameter in the second callback function of the `.then()` method and points to the `reject(somevalue)` line in the Promise constructor.

Promises

Asynchrone actie:

```
let promise = new Promise( (resolve, reject) => {  
  // do something asynchronous  
  // if success  
  //   resolve(somevalue);  
  // otherwise  
  //   reject(somevalue);  
} );
```

```
promise  
  .then(  
    (result) => {console.log(result); }  
  )  
  .catch(  
    (error) => {console.log(error); }  
  )
```

The diagram illustrates the flow of data from the Promise constructor to the then/catch methods. Two arrows originate from the 'resolve' and 'reject' parameters of the Promise constructor's callback function. One arrow points from 'resolve' to the 'result' parameter of the 'then' method's callback. The other arrow points from 'reject' to the 'error' parameter of the 'catch' method's callback.

Voorbeeld1

Faculteit van een getal berekenen. Indien klaar, print het resultaat of een foutmelding.

Uitwerking 1: resolve en reject

In de functie `factorial` wordt een promise aangemaakt en teruggegeven.

In de promise verwijst `resolve` naar de functie die uitgevoerd moet worden bij succes en `reject` naar de functie die uitgevoerd moet worden bij falen.

```

function factorial(number) {
  let promise = new Promise((resolve, reject)=>{
    if(typeof number == 'number' && !isNaN(number)){
      let result=1;
      for (let i=1; i<=number;i++){
        result=result*i;
        if (result==Infinity){
          reject(`${number}! is Infinity`);
          break;
        }
      }
      if(result!=Infinity) {
        resolve(result);
      }
    } else {
      reject(`${number} is not a number`);
    }
  });
  return promise;
}

```

```

factorial(20)
  .then( (result) => {console.log('resolved: ',result);},
        (result) => {console.log('rejected: ',result);}
  );

```


Promises

throw vs reject (throw: rest van de code v. promise niet uitgev.)

```
let promise = new Promise( (resolve) => {  
  // do something asynchronous  
  // if success  
  //   resolve(somevalue);  
  // otherwise  
  //   throw new Error('oops');  
  //   this code not executed  
} );
```

```
promise  
  .then(  
    (result) => {console.log(result); }  
  )  
  .catch(  
    (error) => {console.log(error.message); }  
  )
```

Promises

throw vs reject (throw: rest van de code v. promise niet uitgev.)

```
let promise = new Promise( (resolve) => {  
  // do something asynchronous  
  // if success  
  //   resolve(somevalue);  
  // otherwise  
  //   throw 'oops';  
  //   this code not executed  
} );
```

```
promise  
  .then(  
    (result) => {console.log(result); }  
  )  
  .catch(  
    (error) => {console.log(error); }  
  )
```

```
function factorial(number) {
  let promise = new Promise((resolve)=>{
    if(typeof number == 'number' && !isNaN(number)){
      let result=1;
      for (let i=1; i<=number;i++){
        result=result*i;
        if (result==Infinity){
          throw `${number}! = Infinity`;
        }
      }
      if(result!=Infinity) {
        resolve(result);
      }
    } else {
      throw `${number} is not a number`;
    }
  });
  return promise;
}

factorial(100)
  .then( (result) => {console.log('resolved: ',result);} )
  .catch( (error) => {console.log(error);} );
```

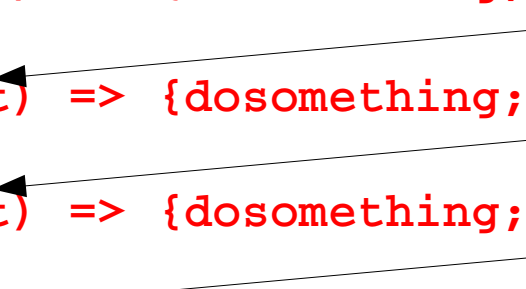


chaining

Meerdere .then's naast elkaar.

promise

```
.then( (result) => {dosomething; return somevalue; } )  
.then( (result) => {dosomething; return somevalue; } )  
.then( (result) => {dosomething; return somevalue; } )  
.then( (result) => {console.log('done: ', result);} )
```

A diagram illustrating the flow of data in a promise chain. Four arrows originate from the 'return somevalue;' line of each .then() call and point to the '(result)' parameter of the subsequent .then() call, showing how the result of one promise becomes the input for the next.

chaining

Meerdere `.then`'s naast elkaar. Optie 1: het resultaat van een promise wordt gechained aan een niet-asynchrone actie.

In het voorbeeld is `factorial` de promise. Na de promise wordt verdubbeld en afgedrukt (dit zijn geen promises).

```
factorial(100)
  .then( (result) => result*2 )
  .then( (result) => {console.log('resolved: ', result);} )
  .catch( (error) => {console.log('rejected: ', error);} );
```

chaining

Meerdere .then's naast elkaar. Optie 2: het resultaat van een promise wordt gechained aan een promise

```
function double(number){
  let promise = new Promise((resolve,reject)=>{
    if(typeof number == 'number' && !isNaN(number)){
      let result=2 * number;
      if(result!=Infinity) {
        resolve(result);
      }
    } else {
      reject(`${number} is not a number`);
    }
  });
  return promise;
}
```

```
factorial(20)
  .then( (result) => double(result) )
  .then( (result) => {console.log('resolved: ', result);} )
  .catch( (error) => {console.log('rejected: ', error);} );
```

chaining

Voorbeeld 2: Optie 2: het resultaat van een promise wordt gechained aan een promise

```
let isMomHappy=true;

function willGetNewPhone() {
  let promise = new Promise(
    function(resolve, reject){
      if(isMomHappy){
        let phone= {
          brand: 'Samsung',
          color:'black'
        };
        resolve(phone);
      }else{
        let reason = new Error('mom is not happy');
        reject(reason);
      }
    }
  );
  return promise;
}
```

chaining

Voorbeeld 2: Optie 2: het resultaat van een promise wordt gechained aan een promise

```
function showOff (phone) {  
  return new Promise(  
    function (resolve, reject) {  
      let message =  
        `I have a new ${phone.color} ${phone.brand} phone`;  
      resolve(message);  
    }  
  );  
};
```

```
willGetNewPhone()  
  .then( (result)=>showOff(result) )  
  .then( (result)=>{console.log(result);} )  
  .catch( (error)=>{console.log(error.message);} );
```



HTTP: GET, POST, PUT, ...

REST: client server architectuur

GET: haal een resource op

GET /persons/

POST: plaats een resource

POST /persons/

request-body: {"name":"tim"}

niet idempotent: eerste keer uitvoeren: id = 1, name="tim" in databank,
2e keer id = 2, name="tim" in databank, ...

PUT: plaats of wijzig een resource

PUT /persons/1

request-body: {"name":"tim"}

idempotent: een of meerdere keren uitvoeren, resultaat is hetzelfde
id = 1, name="tim" in databank

DELETE: verwijder een resource

DELETE /persons/1

...



Fetch API

Verstuur een GET, POST, PUT, DELETE request naar een server met REST-API

De server stuurt een response met JSON in response-body.

Server: json-server (Fake REST-API)

installeer npm (normaal heb je node.js al geïnstalleerd en hoef je niets te doen)
<https://www.npmjs.com/get-npm>

voer

```
npm install -g json-server
```

uit in de prompt.

Plaats het bestand db.json (volgende slide) in een directory.

Voer

```
json-server --watch db.json
```

uit in de prompt (in deze directory).

Fetch API

```
{  
  "persons": [  
    {  
      "id": 1,  
      "name": "sofie"  
    },  
    {  
      "id": 2,  
      "name": "tim"  
    }  
  ]  
}
```

REST-API

GET http://localhost:3000/persons

GET http://localhost:3000/persons/1

POST http://localhost:3000/persons

`{"name": "geert"}`

← in request-body



Fetch API

GET `http://localhost:3000/persons`

get all persons

get person id:

post name:

1 sofie

2 tim

GET `http://localhost:3000/persons/1`

get all persons

get person id:

post name:

1 sofie

POST `http://localhost:3000/persons`
`{"name": "geert"}`

get all persons

get person id:

post name:

3 geert

in request-body

Fetch: GET

```
let url = 'http://localhost:3000/persons/' ;
let output = document.getElementById("div_output");
makeElementEmpty(output);
fetch(url)
  .then((response) =>{
    if(response.status==200){
      return response.json();
    } else {
      throw `error with status ${response.status}`;
    }
  })
  .then((persons) => {
    let data = [];
    for (let person of persons) {
      data.push([person.id, person.name]);
    }
    let table=makeTable(data);
    output.appendChild(table);
  })
  .catch( (error) => {
    output.appendChild(document.createTextNode(error));
  } );
```

*verstuur GET request
indien status 200:
(gelukt!) json-string in response-body
wordt geparsed naar javascript code
anders werp een error op*

Fetch: GET

```
let url = 'http://localhost:3000/persons/' ;
let output = document.getElementById("div_output");
makeElementEmpty(output);
fetch(url)
  .then((response) =>{
    if(response.status==200){
      return response.json();
    } else {
      throw `error with status ${response.status}`;
    }
  })
  .then((persons) => {
    let data = [];
    for (let person of persons) {
      data.push([person.id, person.name]);
    }
    let table=makeTable(data);
    output.appendChild(table);
  })
  .catch( (error) => {
    output.appendChild(document.createTextNode(error));
  } );
```

*doorloop de array
persons, maak er een 2D
matrix van en plaats de
matrix in een <table>*

Fetch: GET

```
let url = 'http://localhost:3000/persons/' ;
let output = document.getElementById("div_output");
makeElementEmpty(output);
fetch(url)
  .then((response) =>{
    if(response.status==200){
      return response.json();
    } else {
      throw `error with status ${response.status}`;
    }
  })
  .then((persons) => {
    let data = [];
    for (let person of persons) {
      data.push([person.id, person.name]);
    }
    let table=makeTable(data);
    output.appendChild(table);
  })
  .catch( (error) => {
    output.appendChild(document.createTextNode(error));
  } );
```

foutmelding

Fetch: POST

```
let url = 'http://localhost:3000/persons/';
let output = document.getElementById("div_output");
let name = document.getElementById("txt_name").value;
let person = {name: name};
makeElementEmpty(output);
fetch(url,
  {
    method: "POST",
    body: JSON.stringify(person),
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    }
  })
.then((response) => {
  if (response.status == 201) {
    return response.json();
  } else {
    throw `error with status ${response.status}`;
  }
})
```

*verstuur POST request
met json in request-body
{"name":"geert"}*

Fetch: POST

```
let url = 'http://localhost:3000/persons/';
let output = document.getElementById("div_output");
let name = document.getElementById("txt_name").value;
let person = {name: name};
makeElementEmpty(output);
fetch(url,
  {
    method: "POST",
    body: JSON.stringify(person),
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    }
  })
  .then((response) => {
    if (response.status == 201) {
      return response.json();
    } else {
      throw `error with status ${response.status}`;
    }
  })
```

*201: resource created
(gelukt!)
zet de json-string in de
response-body om naar
javascript
anders: werp error op*

Fetch: POST

```
.then((person) => {  
    let data = [];  
    data.push([person.id, person.name]);  
    let table=makeTable(data);  
    output.appendChild(table);  
})  
.catch((error) => {  
    output.appendChild(  
        document.createTextNode(error));  
});
```

*toon het resultaat in een
<table>*

Fetch: POST

```
.then((person) => {  
  let data = [];  
  data.push([person.id, person.name]);  
  let table=makeTable(data);  
  output.appendChild(table);  
})  
.catch((error) => {  
  output.appendChild(  
    document.createTextNode(error));  
});
```

toon de foutmelding

Voorbeeld2

get id's id:

Id 2 heeft name tim. De name tim komt voor op de volgende id's: 2 3 4

```
persons.json x
1|{
2  "persons": [
3    {
4      "id": 1,
5      "name": "jan"
6    },
7    {
8      "id": 2,
9      "name": "tim"
10   },
11   {
12     "id": 3,
13     "name": "tim"
14   },
15   {
16     "id": 4,
17     "name": "tim"
18   }
19 ]
20 }
21 }
```

```
8 function handleGetIds() {
9     let url = 'http://localhost:3000/persons/';
10    let id = document.getElementById("txt_id").value;
11    let output = document.getElementById("div_output");
12    makeElementEmpty(output);
13    fetch(`${url}${id}`)
14        .then((response) => {
15            if (response.status == 200) {
16                return response.json();
17            } else {
18                throw `error with status ${response.status}`;
19            }
20        })
21        .then((person) => {
22            let name = person.name;
23            output.appendChild(document.createTextNode(
24                `Id ${id} heeft name ${name}. De name ${name} komt voor op de volgende id's: `));
25            return fetch( `${url}?name=${name}`);
26        })
27        .then((response) => {
28            if (response.status == 200) {
29                return response.json();
30            } else {
31                throw `error with status ${response.status}`;
32            }
33        })
34        .then((persons) => {
35            for(let person of persons){
36                let id = person.id;
37                let name = person.name;
38                output.appendChild(document.createTextNode(` ${id}`));
39            }
40        })
41        .catch((error) => {
42            output.appendChild(document.createTextNode(error));
43        });
}
```