

# **AJAX IN PLAIN JAVASCRIPT**

- Was sind Callbacks?
- Wie funktioniert Synchroner und Asynchroner Code in Javascript?
- Wie rufe ich in Javascript Daten von einem Server ab?
- Wie machen uns Promises das Leben leichter?
- Was ist async/await?



*“In computer programming, a callback is a piece of executable code that is passed as an argument to other code, which is expected to call back (execute) the argument at some convenient time. The invocation may be immediate as in a synchronous callback or it might happen at later time, as in an asynchronous callback.”*

Wikipedia - Callback (computer programming)

```
1 function someFunction(callback) {
2     console.log('do something');
3
4     callback()
5 }
6
7 function someCallbackFunction() {
8     console.log('callback called');
9 }
10
11 function onClick() {
12     someFunction(someCallbackFunction)
13 }
```

Code aus dem Repo auschecken:

[gitlab.com/dhbw\\_webengineering\\_2/plain\\_javascript](https://gitlab.com/dhbw_webengineering_2/plain_javascript)

Branch: "step\_1"

```
1 function someFunction(callback) {
2     console.log('do something');
3
4     callback()
5 }
6
7
8 function onClick() {
9     someFunction(() => console.log('callback called'));
10 }
```

```
1 function someFunction(callback) {
2     console.log('do something');
3
4     callback('callback called');
5 }
6
7 function someCallbackFunction(string) {
8     console.log(string);
9 }
10
11 function onClick() {
12     someFunction(someCallbackFunction);
13 }
```



```
1 function someFunction(callback) {  
2     console.log('do something');  
3  
4     callback('callback called');  
5 }  
6  
7 function onClick() {  
8     someFunction((string) => console.log(string));  
9 }
```



- Programmabschnitte werden der Reihe nach ausgeführt
- im Code ist bereits festgehalten wie diese Reihenfolge aussieht
- zur Laufzeit kann sich diese nicht ändern

- Programmschritte werden nicht Synchron ausgeführt
- es muss nicht auf Programmschritte gewartet werden
- Programmschritte können gleichzeitig ausgeführt werden (nicht in JS)

```
1 function synchron() {
2     console.log("step 1");
3     console.log("step 2");
4     console.log("step 3");
5 }
6
7 function asynchron() {
8     console.log("step 1");
9     setTimeout(() => {
10         console.log("step 2");
11     }, 1000);
12     console.log("step 3");
13 }
```

Branch: "step\_2"



```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```



```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```

```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```

```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```

```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```

```
1 function showDadJoke() {
2     const xhr = new XMLHttpRequest();
3     xhr.onload = function () {
4         const joke = JSON.parse(xhr.response);
5         document.getElementById('joke').innerHTML = joke.jc
6     }
7     xhr.open('GET', 'https://icanhazdadjoke.com/', true);
8     xhr.setRequestHeader('Accept', 'application/json');
9     xhr.send();
10 }
```

Branch: "step\_3"

- Schickt mehrere Backendrequests nacheinander ab
- Falls ihr nicht weiter wisst: Branch "step\_4"
- Wie kann ich die Requests parallel abschicken, um Zeit zu sparen?

- Schaut in die Netzwerkkonsole eures Browsers, wenn ihr mit den verschiedenen Calls testet
- Hier bekommt ihr angezeigt wie lange der Request dauert
- Ihr könnt außerdem sehen, ob sie nacheinander oder parallel abgeschickt werden



Wie muss der Code aussehen, wenn wir sie gleichzeitig senden wollen, wir aber den HTML Code erst anpassen wollen, wenn beide Witze zurückgekommen sind?



- erleichtern uns das Arbeiten mit asynchronem Code
  - sie machen aus einem Callback wieder einen "regulären" Rückgabewert
  - dieser Rückgabewert ist Zeitversetzt
  - es wird ein "Versprechen" gegeben, dass ein Wert zurückgegeben wird

```
1 function callBackend() {  
2     return new Promise();  
3 }
```

```
1 const promise = callBackend();  
2 promise  
3     .then((result) => console.log(result))  
4     .catch((error) => console.error(error));
```

- Wichtig: an dem XMLHttpRequest ändert sich erstmal nichts
- der Callback muss also in ein Promise umgewandelt werden

```
1 function get() {
2     return new Promise((resolve, reject) => {
3         const xhr = new XMLHttpRequest();
4         xhr.onload = () => {
5             if (xhr.status >= 200 && xhr.status < 300) {
6                 resolve(JSON.parse(xhr.response));
7             } else {
8                 reject(xhr.statusText);
9             }
10        }
11    });
12 }
```

- Wichtig: an dem XMLHttpRequest ändert sich erstmal nichts
- der Callback muss also in ein Promise umgewandelt werden

```
1 function get() {
2     return new Promise((resolve, reject) => {
3         const xhr = new XMLHttpRequest();
4         xhr.onload = () => {
5             if (xhr.status >= 200 && xhr.status < 300) {
6                 resolve(JSON.parse(xhr.response));
7             } else {
8                 reject(xhr.statusText);
9             }
10        }
11    });
12 }
```

- Wichtig: an dem XMLHttpRequest ändert sich erstmal nichts
- der Callback muss also in ein Promise umgewandelt werden

```
1 function get() {
2     return new Promise((resolve, reject) => {
3         const xhr = new XMLHttpRequest();
4         xhr.onload = () => {
5             if (xhr.status >= 200 && xhr.status < 300) {
6                 resolve(JSON.parse(xhr.response));
7             } else {
8                 reject(xhr.statusText);
9             }
10        }
11    });
12 }
```

```
1 a((resultFromA) => {
2     b(resultFromA, (resultFromB) => {
3         c(resultFromB, (resultFromC) => {
4             d(resultFromC, (resultFromD) => {
5                 console.log(resultFromD);
6             });
7         });
8     });
9 });
```

```
1 a().then((result) => {
2     return b(result);
3 }).then((result) => {
4     return c(result);
5 }).then((result) => {
6     return d(result);
7 }).then((result) => {
8     console.log(result);
9 });
```



- Promise.all()
- Promise.allSettled()
- Promise.any()
- ...

```
1 Promise.all([get(url), get(url)]).then(([joke1, joke2]) => {  
2     document.getElementById('joke1').innerHTML = joke1.joke;  
3     document.getElementById('joke2').innerHTML = joke2.joke;  
4 }).catch((error) => console.log(error));
```

- Callback Hell wird vermieden
- Fehler werden expliziter behandelt
- Promises Methoden helfen bei asynchronem Code

- Branch: "step\_5" - ein Beispiel für Promises
- Vermeidet die kleine "callback hell" in Branch "step\_5"
- Aufgabe von vorhin (Paralleler Request HTML editieren erst nach dem resollen beider Requests) jetzt mit Promises
- Branch: "step\_6" - ein Beispiel für Promise.all() (Lösung)



- `async/await` macht Promises noch schöner
- mit `async` können wir Funktionen markieren, die asynchronen Code enthalten
- mit `await` warten wir auf ein Ergebnis eines Promises
- mit Codebeispielen versteht man es einfacher

```
1 async function showDadJoke() {  
2     const url = 'https://icanhazdadjoke.com/';  
3  
4     const joke1 = await get(url);  
5     const joke2 = await get(url);  
6  
7     document.getElementById('joke1').innerHTML = joke1.joke;  
8     document.getElementById('joke2').innerHTML = joke2.joke;  
9 }
```

```
1 async function showDadJoke() {  
2     const url = 'https://icanhazdadjoke.com/';  
3  
4     const joke1 = await get(url);  
5     const joke2 = await get(url);  
6  
7     document.getElementById('joke1').innerHTML = joke1.joke;  
8     document.getElementById('joke2').innerHTML = joke2.joke;  
9 }
```

```
1 async function showDadJoke() {  
2     const url = 'https://icanhazdadjoke.com/';  
3  
4     const joke1 = await get(url);  
5     const joke2 = await get(url);  
6  
7     document.getElementById('joke1').innerHTML = joke1.joke;  
8     document.getElementById('joke2').innerHTML = joke2.joke;  
9 }
```



```
1 async function showDadJoke() {  
2     const url = 'https://icanhazdadjoke.com/';  
3  
4     const joke1 = await get(url);  
5     const joke2 = await get(url);  
6  
7     document.getElementById('joke1').innerHTML = joke1.joke;  
8     document.getElementById('joke2').innerHTML = joke2.joke;  
9 }
```

- Branch: "step\_7" - enthält ein Beispiel für async/await
- Werden die Calls jetzt nacheinander abgeschickt?
- Wie bekommen wir sie wieder parallel?
- Lösung gibt es auch Branch: "step\_8"
- Nehmt euch auch hier gerne die Zeit und spielt etwas damit herum.