AJAX IN PLAIN JAVASCRIPT

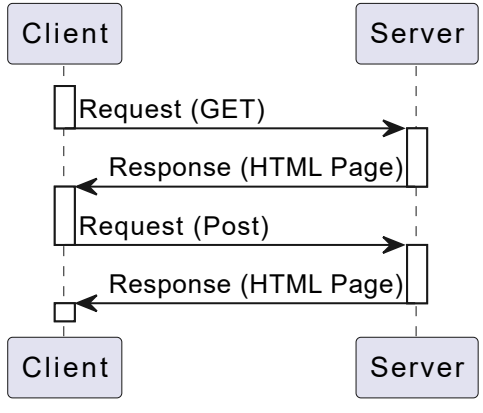
LERNZIELE

- Was ist Ajax?
- 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?

AJAX

(Asynchronus JavaScript and XML)

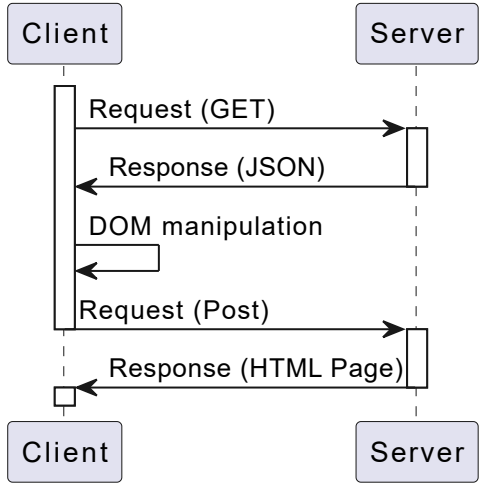
MULTI PAGE APPLICATION (ERINNERUNG)



WAS IST AJAX?

- es ist keine Programmiersprache!
- es handelt sich um ein Programmierprinzip
- Daten anzeigen, ohne eine neue HTML Seite auszuliefern

AJAX


Client
Server
Request (GET)
Response (JSON)
DOM manipulation
Request (Post)
Response (HTML Page)
Client
Server

CALLBACKS

WAS SIND CALLBACKS?

“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)

BEISPIEL CALLBACK:

```
 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 onButtonClick() {
12     someFunction(someCallbackFunction)
13 }
```

PRAXIS

Code aus dem Repo auschecken:

gitlab.com/dhbw_webengineering_2/plain_javascript

Branch: "step_1"

BEISPIEL CALLBACK ANONYM:

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

BEISPIEL CALLBACK MIT PARAMETER:

```
 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 onButtonClick() {
12     someFunction(someCallbackFunction);
13 }
```

BEISPIEL CALLBACK MIT PARAMETER ANONYM:

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

PROMISES

PROMISES

- 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

PROMISES VERWENDUNG:

```
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));
```

PROMISES BEISPIEL:

- 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 }
```

PROMISES BEISPIEL:

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

```
function get() {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.onload = () => {
            if (xhr.status >= 200 && xhr.status < 300) {
                resolve(JSON.parse(xhr.response));
            } else {
                reject(xhr.statusText);
            }
        }
    });
}
```

PROMISES BEISPIEL:

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

```
function get() {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.onload = () => {
            if (xhr.status >= 200 && xhr.status < 300) {
                resolve(JSON.parse(xhr.response));
            } else {
                reject(xhr.statusText);
            }
        }
    });
}
```

CALLBACK HELL

```
a((resultFromA) => {
    b(resultFromA, (resultFromB) => {
        c(resultFromB, (resultFromC) => {
            d(resultFromC, (resultFromD) => {
                console.log(resultFromD);
            });
        });
    });
});
```

```
a().then((result) => {
    return b(result);
}).then((result) => {
    return c(result);
}).then((result) => {
    return d(result);
}).then((result) => {
    console.log(result);
});
```

PROMISES METHODEN

- 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));
```

VORTEILE VON PROMISES

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

PRAXIS

- 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 resolven beider Requests) jetzt mit Promises
- Branch: "step_6" - ein Beispiel für Promise.all() (Lösung)

ASYNC/AWAIT

SYNTAKTISCHER ZUCKER

- asnyc/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

CODEBEISPIEL

```
async function showDadJoke() {
    const url = 'https://icanhazdadjoke.com/';

    const joke1 = await get(url);
    const joke2 = await get(url);

    document.getElementById('joke1').innerHTML = joke1.joke;
    document.getElementById('joke2').innerHTML = joke2.joke;
}
```

CODEBEISPIEL

```
async function showDadJoke() {
    const url = 'https://icanhazdadjoke.com/';

    const joke1 = await get(url);
    const joke2 = await get(url);

    document.getElementById('joke1').innerHTML = joke1.joke;
    document.getElementById('joke2').innerHTML = joke2.joke;
}
```

CODEBEISPIEL

```
async function showDadJoke() {
    const url = 'https://icanhazdadjoke.com/';

    const joke1 = await get(url);
    const joke2 = await get(url);

    document.getElementById('joke1').innerHTML = joke1.joke;
    document.getElementById('joke2').innerHTML = joke2.joke;
}
```

CODEBEISPIEL

```
async function showDadJoke() {
    const url = 'https://icanhazdadjoke.com/';

    const joke1 = await get(url);
    const joke2 = await get(url);

    document.getElementById('joke1').innerHTML = joke1.joke;
    document.getElementById('joke2').innerHTML = joke2.joke;
}
```

PRAXIS

- Branch: "step_7" - enthält ein Beispiel für async/await
- Werden die Calls jetzt nacheinander abgeschickt?
- Wie bekommen wir sie wieder parallel?
- Läsung gibt es auch Branch: "step_8"
- Nehmt euch auch hier gerne die Zeit und spielt etwas damit herum.