

Tutorium zu Kapitel WT:V

- ❑ DOM-Manipulation
- ❑ `EventListener` & Callbacks
- ❑ AJAX & `XMLHttpRequest`

DOM-Manipulation

Elemente erzeugen, manipulieren und löschen

- ❑ Element erzeugen:

```
let newElement = document.createElement("tag-name")
```

- ❑ Element ins DOM einhängen:

```
document.body.appendChild(element)
```

- ❑ Element löschen:

```
newElement.remove()
```

- ❑ HTML-Klasse setzen:

```
newElement.classList.add("class-name")
```

- ❑ HTML-Klasse entfernen:

```
newElement.classList.remove("class-name")
```

DOM-Manipulation

`querySelector`, `querySelectorAll`

- ❑ Bestimmtes Element im DOM finden:

```
let element = document.querySelector(cssSelector)
```

Beispiel:

```
let button = document.querySelector("button")
```

- ❑ Liste ([NodeList](#)) von Elementen im DOM finden:

```
let elements = document.querySelectorAll(cssSelector)
```

Beispiel:

```
let anchors = document.querySelectorAll("a")
```

EventListener & Callbacks

Theorie

- ❑ `EventListener` ermöglichen es, auf bestimmte Ereignisse (Events) zu reagieren.

```
target.addEventListener(type, callback);
```

		Callback-Funktion, mit der
		auf das Event reagiert wird
		Event-Typ, auf den reagiert werden soll

Auf diesem Event-Ziel (i.d.R. ein DOM-Knoten) wird der `EventListener` registriert und horcht fortan auf Ereignisse eines bestimmten Typs.

- ❑ `callback` ist dabei eine Referenz auf eine Funktion, die immer dann aufgerufen wird, wenn das Event ausgelöst wird

EventListener & Callbacks

Beispiel

❑ Variante 1: Benannte Funktion als Callback

```
function removeElement(event) {  
    event.target.remove();  
}
```

```
document.body.addEventListener("click", removeElement);
```

❑ Variante 2: Anonyme Funktion als Callback

```
document.body.addEventListener("click", function(event) {  
    event.target.remove();  
});
```

❑ Variante 3: Anonyme Arrow-Funktion als Callback (ES6)

```
document.body.addEventListener("click", (event) => {  
    event.target.remove();  
});
```

EventListener & Callbacks

Anmerkungen

- ❑ Grober Ablauf:
 1. Innerhalb des Event-Ziels `document.body` (das HTML `body`-Element), ...
 2. erwarte das `click`-Event ...
 3. und führe die Callback-Funktion aus.

- ❑ Für das Absenden eines Formulars gibt es das [submit](#)-Event.

AJAX

Asynchrone HTTP-Requests mit `XMLHttpRequest`

- ❑ Standardmäßig wird bei HTTP-Requests eine Ressource im aktuellen Fenster geladen
- ❑ AJAX („Asynchronous JavaScript and XML“) erlaubt das Verarbeiten von Requests, ohne die aktuelle Seite zu verlassen
- ❑ Solche Requests werden zudem asynchron ausgeführt. Sie unterbrechen nicht die Ausführung von synchronem Code.
- ❑ In JavaScript steht für die Umsetzung von AJAX das `XMLHttpRequest`-Objekt zu Verfügung.

AJAX

XMLHttpRequest-Beispiel: Schritt für Schritt

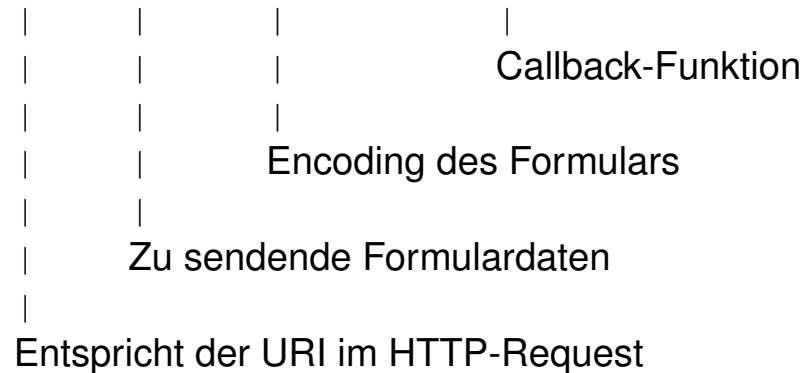
- aus der `feedreader.js` von der Kurswebseite:

```
function postEntry(url, data, encoding, callback) {  
    const req = new XMLHttpRequest();  
    req.open("POST", url);  
    req.setRequestHeader("Content-Type", encoding);  
    req.addEventListener("loadend",  
        createCallbackFunction(req, callback));  
    req.send(data);  
}
```


AJAX

XMLHttpRequest-Beispiel: Schritt für Schritt

❑ **function** `postEntry(url, data, encoding, callback)`



❑ `url`: absolute oder relative Pfadangabe

❑ `data`: Formulardaten in der Form `key1=value (&key2=value) *`

❑ `encoding`: Setzt `Content-Type` des POST-Requests auf den MIME Type des Inhalts. Dieser ist für Formulare mit der Methode `post` Standardmäßig `application/x-www-form-urlencoded`. Mit diesem Content-Type erwartet der HTTP-Server die POST-Daten in der Form `key1=value (&keyN=value) *`

❑ `callback`: Funktion, die die HTTP-Antwort des Servers verarbeiten soll

AJAX

XMLHttpRequest-Beispiel: Schritt für Schritt

- Ein neues XMLHttpRequest-Objekt wird erzeugt

```
const req = new XMLHttpRequest();
```

- Das loadend-Event wird aufgerufen, wenn der Request beendet ist (erfolgreich oder nicht erfolgreich)

```
req.addEventListener("loadend",  
    createCallbackFunction(req, callback));
```

- Die Callback-Funktion soll den HTTP-Status der Server-Antwort prüfen und diese ggf. weiterverarbeiten
- Soll die Callback-Funktion nur bei Antworten mit Status 200 OK aufgerufen werden, kann statt loadend das load-Event verwendet werden.

AJAX

- ❑ Initialisiert ein neues HTTP-Request-Objekt

```
req.open("POST", url);
```

- ❑ Setzt das HTTP-Headerfeld `Content-Type` auf den Wert in `encoding`.

```
req.setRequestHeader("Content-Type", encoding);
```

- ❑ Sendet den HTTP-Request an den Server. Das optionale Argument enthält den Request-Body. Bei asynchronen XMLHttpRequests wartet die Methode nicht auf eine Antwort vom Server sondern gibt sofort zurück.

```
req.send(data);
```

- ❑ Nach dem Aufruf von `req.send` ist die `postEntry`-Funktion abgearbeitet und gibt an den aufrufenden Kontext zurück. Dort kann nun sofort weiterer Code ausgeführt werden, während im Hintergrund der HTTP-Request vom Server bearbeitet wird.

- ❑ Sobald eine Antwort vom Server empfangen wurde, wird die an das `loadend`-Event gebundene Callback-Funktion aufgerufen.