Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Web APIs

Formulare validieren

Attribute, Constraint Validation API

Form Elemente - Rückblick

Input Elemente und Buttons

- Input-Types:
  - hidden
  - text, search
  - tel, url, email
  - password
  - number, range
  - checkbox, radio
  - file
  - submit, image, reset
  - button
  - Nur zum Teil unterstützt:
    - date, month, week, time, datetime-local
    - color
- button (types: reset, submit, button)
- select, option, optgroup
- textarea

Strukturierung und Information

- form
  - Für den Upload von Files muss das enctype-Attribut auf "`multipart/form-data`" gesetzt sein
- label
  - Klick auf Label aktiviert entsprechendes Element
- fieldset, legend
  - Gruppierung
- Nur zum Teil unterstützt:
  - datalist (partial Support Safari und IE)
  - progress (iOS Safari nicht zu 100%)
  - output (ohne IE)
  - meter (ohne IE)

Validation

Element-Attribute

- Über den “richtigen” Input-**type**
- **required** (verhindert ein Submit)

- **max** / **min** / **step** (Numeric oder Date/Time)
- **maxlength** (Text)

- Für text-, search-, tel-, url-, email- und password-Typen:
  - **pattern** (regulärer Ausdruck, Beispiele oder Testen)

- Nur für den file-input-Type:
  - **accept** (z.B. “`.png`” oder “`text/*`”)

CSS Pseudoklassen

- **:valid** / **:invalid**
- **:required** / **:optional**
- **:in-range** / **:out-of-range** (ohne IE)

- Die Pseudoklassen werden während der Eingabe überprüft und entsprechend angewendet

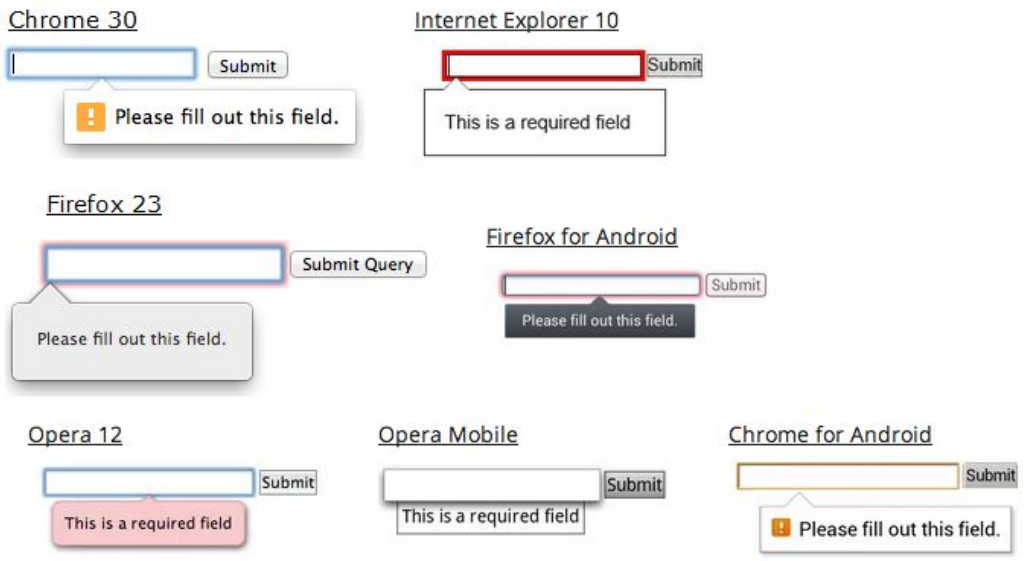
- Mehrere Pseudoklassen können kombiniert werden, z.B.:

```
input:required:invalid {
  border-color: red;
}
```

https://www.tjvantoll.com/speaking/slides/constraint-validation/chicago/#/39

Bitte beachten

- Validierung im Browser ersetzt nie eine Validierung auf dem Server!
- Je nach Browser werden die Validierungen unterschiedlich dargestellt…



https://developer.mozilla.org/de/docs/Web/HTML/Element/Input
https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete

Weitere nützliche Attribute

- **title**: Sichtbarer Text bei einem Hover mit der Maus
- **placeholder**: Sichtbarer Text innerhalb des Feldes welcher bei einer Eingabe automatisch verschwindet
- **autocomplete**: `on`, `off`, `name`, `current-password` **usw.**
  Die Browser halten sich nicht immer daran, z.B. für Adressen oder Passwörter.
- **spellcheck**: `true` oder `false`, mehr als Hinweis an den Browser, verhindert nicht ein Submit bei Fehlern… :-)
- **autofocus**: Das ausgewählte Element erhält beim Laden der Seite den Fokus (z.B. Benutzername bei einem Login-Formular)
- **readonly**: Nur Lesen, Elemente sehen jedoch gleich aus
- **disabled**: Visuell sichtbar, Werte werden jedoch nicht via Submit mitgeschickt

https://html.spec.whatwg.org/multipage/forms.html#the-constraint-validation-api
https://www.html5rocks.com/en/tutorials/forms/constraintvalidation/
https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation
HTML5-Beispiele demo_formular_02.html und demo_formular_03.html

Constraint Validation API

Möglichkeit, die Validierung mit JavaScript zu beeinflussen

- Methoden:
  - **checkValidity()**: Ein Element oder das ganze Formular überprüfen, bei einem Fehler wird `false` zurückgegeben und der invalid-Event ausgelöst.
  - **reportValidity()**: Wie checkValidity, jedoch meldet der Browser einen Fehler direkt dem User. Bei einem Fehler wird also ein invalid-Event ausgelöst und der Browser behandelt den Fehler wie wenn das Formular abgeschickt wird.
  - **setCustomValidity**(message): Setzen einer eigenen Nachricht welche der Browser bei einem Fehler anzeigt und setzt das Element auf "invalid". Ein leerer String entfernt die Nachricht und setzt das Element auf "valid". D.h. die Fehlermeldungen können damit nicht im Voraus definiert werden, da jeder Aufruf mit einer Fehlermeldung einen Fehler auslöst.
- Eigenschaften (readonly):
  - **willValidate**: `true` falls das Element überprüft werden kann, sonst `false` (z.B. ein Form-Element mit disabled-Attribut).
  - **validationMessage**: Nachricht, welche der Browser bei einem Fehler anzeigt.
  - **validity**: Objekt mit Details zum Status als Boolean-Eigenschaften: `valueMissing, typeMismatch, patternMismatch, tooLong, tooShort, rangeUnderflow, rangeOverflow, stepMismatch, badInput, customError, valid.`
- Events:
  - **invalid**: Wird bei jedem negativen Check ausgelöst
  - **change** oder **keypress** oder **keyup** oder **input**: Für positive Checks gibt es keinen Event, daher bei jeder Änderung die Validität abfangen (via `validity.valid` -Eigenschaft) oder neu Prüfen
  - Unterschiede der Events, Fazit: input-Event für input- und textarea-Felder, change (oder input) für select-, checkbox- und radio-Elemente

Form Validation (3 Punkte)

- Auf der Profilseite soll mit geeigneten Formularfeld-Attributen:
  - Die Felder "Email", "Country" und "Profil Picture" als required markiert werden.
  - Das Feld "ZIP Code" darf nur Zahlen enthalten (via input-type oder pattern-Attribut).
- Auf der Index-Seite (login) soll mit der Constraint Validation API eine eigene Fehlermeldung angezeigt werden, falls kein Nickname eingegeben wurde (also der Default-Text des Browser soll mit einem eigenen Text ersetzt werden).

Daten abrufen und senden

Fetch, FormData und File API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
https://caniuse.com/#feat=fetch
https://developer.mozilla.org/en-US/docs/Web/API/AbortController/abort

Fetch API

- XMLHttpRequest mit Promises ohne IE Support
- Experimenteller Support für den Abbruch von Anfragen via `AbortController` und `AbortSignal`
- Methoden / Klassen:
  - `fetch()`
  - `Headers`
  - `Request` und `Response`
  - Body mixins:
    - `Body.json()`
    - `Body.blob()`
    - …

Plain JavaScript

```
fetch('http://example.com/movies.json')
 .then(response => response.json())
 .then(json => {
   console.log(json)
 })
```

https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch
HTML5-Beispiele demo_fetch_01.html, demo_fetch_02.html und demo_fetch_03.html

Fetch Methode

Syntax: `fetchResponsePromise = fetch(url, options);`

- Mögliche Optionen sind:
  - **method**: z.B. GET oder POST
  - **headers**: Zusätzliche HTTP Headers (nicht alle sind erlaubt)
  - **body**: Senden von Daten
  - **mode**: cors (default), no-cors (keinen Zugriff auf response Daten) oder same-origin
  - **credentials**: Senden von Cookies einschalten
  - **cache**: Steuern des Cache-Verhaltens
  - **redirect**: Weiterleitungen folgen (per default an)
  - **referrer** / referrerPolicy: Steuern des Referrer Header (Herkunft)
  - **integrity**: Hash zum Überprüfen der Daten (SRI)
  - **keepalive**: Verbindung auch nach dem "Unload" einer Seite offen lassen (z.B. zum Senden von Tracking Code analog sendBeacon, kein Zusammenhang mit dem HTTP Keepalive header)
  - **signal**: Zum Abbrechen einer Verbindung

https://developer.mozilla.org/de/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest
https://www.html5rocks.com/en/tutorials/file/xhr2/

Daten senden

Neben Text (USVString), kann der Fetch-Body auch folgendes sein:

- **Blob / File**: Unveränderbare Daten, ein File ist ein Blob mit spezifischen Eigenschaften
- **ArrayBuffer (BufferSource)**: Raw Daten, jedoch veränderbar mit Hilfe von DataViews
- **FormData**: Interface für Schlüssel/Werte-Paare von Formulardaten
  - Sendet Daten wie ein Formular mit dem encoding-Typ "multipart/form-data" (File-Upload)
- **URLSearchParams**: Interface zum einfachen Senden von Parametern wie bei einem GET-Request (z.B. name=Foobar&year=2019)

Beispiel mit JSON-Daten

```
fetch('http://example.com/new', {
 method: 'POST',
 headers: {
   Accept: 'application/json',
   'Content-Type': 'application/json'
 },
 body: JSON.stringify({
   name: 'Foobar',
   year: 2019
   // ...
 })
})
 .then(
   //...
 )
```

https://developer.mozilla.org/de/docs/Web/API/FormData
https://www.html5rocks.com/en/tutorials/file/xhr2/
HTML5-Beispiele demo_formular_02.html und demo_formular_03.html

FormData Interface

Methoden

- append(name, string)
- append(name, Blob/File, filename)

- Weitere, noch nicht überall unterstützte Methoden:
  - get() / getAll()
  - set()
  - delete()
  - has()
  - entries()
  - keys() / values()

Beispiele

```
 <!-- Ein bestehendes Formular anreichern -->
<form id="myForm" name="myForm">
 <input type="text" id="textfield" name="textfield" />
 <input type="file" id="userfile" name="userfile" />
</form>
<script>
 // FormData mit bestehendem Formular füllen
 var myForm = document.getElementById('myForm')
 var formData = new FormData(myForm)
 formData.append('secret_token', 'abc')
 // XMLHttpRequest oder fetch vorbereiten und senden ...
</script>
```

```
 <!-- Oder ohne Formular -->
<script>
 var formData = new FormData()
 formData.append('textfield', 'abc')
 formData.append('userfile', File - Object, 'textfile27.txt')
 // XMLHttpRequest oder fetch vorbereiten und senden ...
</script>
```

File API

Übersicht

- Möglichkeit um mit JavaScript auf Files zuzugreifen
- Normalerweise via FileList von File-input-Elementen oder via Drag & Drop
- Ein File ist ein spezifischer Blob, d.h. er kann gelesen und manipuliert werden, via XMLHttpRequest verschickt oder auch neu erstellt werden

File Interface

- **File**(bits, name[, options])
- File.**lastModified**
- File.**name**
- File.**size**
- File.**type**
- Lesen geht via FileReader Interface, z.B.: `FileReader.readAsArrayBuffer()` oder `FileReader.readAsText()`
- Mit der statischen Methode `URL.createObjectURL(file or blob)` kann z.B. via img-Element ein im Browser verändertes Bild dargestellt werden

AJAX (5 Punkte)

Kann mit XMLHttpRequest oder der Fetch API gelöst werden.

Bitte die public API unter https://chat.humbapa.ch/api benutzen. Eine Übersicht der verfügbaren Endpunkte ist unter https://chat.humbapa.ch/help/ zu finden. Falls was nicht geht bitte melden…

- Beim Login soll ein neuer User erstellt werden oder falls im Browser bereits ein abgespeicherter User existiert, dieser aktualisiert werden (nickname und/oder status via PUT Request).
- Beim Aufrufen der Chat-Seite sollen alle User (inkl. Status) und alle bisherigen Nachrichten (inkl. Timestamp) geholt und dargestellt werden.
- Das User-Profil (Avatar) soll entsprechend dargestellt werden. Die Chat-API liefert dazu im Feld "avatar" einfach eine Zahl welche entsprechend in eine Bild-Url umgewandelt werden muss. Wo der Avatar angezeigt wird, spielt keine Rolle (also entweder bei der Nachricht oder bei der User-Liste oder an beiden Orten).

Storage APIs

Web Storage API und IndexedDB

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Web Storage API

Übersicht

- Einfacher Key/Value Store
  - Nur Strings
- Zwei Arten:
  - sessionStorage
  - localStorage
- Zugriff erfolgt synchron
- Browser Support ist sehr gut
- Grösse ist allerdings limitiert, z.B.:
  - Chrome und Firefox: 5MB
  - IE: 10MB

Anwendung

▶️ Zugriff erfolgt über das Storage Interface:

- setItem(key, value)
- getItem(key)
- removeItem(key)

```
// Speichern im Session-Storage:
sessionStorage.setItem('key', 'value')
// oder im Local-Storage:
localStorage.setItem('key', 'value')

// Daten holen
var data = localStorage.getItem('key')
if (data === null) {
 // Keine Daten mit diesem Key
}
```

https://developer.mozilla.org/en-US/docs/Web/API/StorageEvent
https://developer.mozilla.org/en-US/docs/Web/API/Storage

Web Storage API

Weitere Hinweise

- Alle Webseiten der gleichen Domain/Protokoll/Port haben Zugriff auf den gleichen Web Storage
- Im Vergleich: Bei Cookies kann der Pfad zusätzlich eingeschränkt werden (dafür werden die Daten bei jedem Seitenaufruf übertragen)
- Zugriff im Inkognito Modus (und bei lokalen Files) wird je nach Browser unterschiedlich gehandhabt
- Objekte können auch als serialisiertes JSON gespeichert werden (`JSON.stringify` und `JSON.parse`)
- Änderungen am Storage können mit dem storage-Event in den anderen Tabs/Fenster erkannt werden
- Weitere Eigenschaften/Methoden des Storage Interfaces:
  - Storage.clear()
  - Storage.key(index)
  - Storage.length

IndexedDB

Überblick

- Möglichkeit grössere Datenmengen zu speichern, wie viel ist je nach Browser unterschiedlich
- Zugriff ist wie beim Web Storage von allen Seiten der gleichen Domain und Protokoll möglich (Same Origin Policy)
- Zugriff im Inkognito Modus ist ebenfalls nicht einheitlich geregelt
- Objektorientierter Key/Value Store (keine relationale Datenbank)
- Zugriff auf die “Datenbank” erfolgt asynchron und immer im Kontext einer Transaktion
- Datenbanken haben eine Version
- Eigentlich eine alternative zu WebSQL, diese Variante wurde jedoch vom W3C verworfen da die Browser als Backend nur Sqlite implementierten…

Grundschema

Ablauf

1. *"Öffne eine Datenbank und starte eine Transaktion.*
2. *Erzeuge einen Objektspeicher.*
3. *Fordere die Ausführung von Datenbankoperationen an, wie das Hinzufügen und Auslesen von Daten.*
4. *Warte auf die richtige Art von DOM-Ereignis welche auftritt, wenn die Operation beendet ist.*
5. *Verarbeite die Ergebnisse (, welche im Anforderungsobjekt gefunden werden können).*"

Hinweise

- Zugriff auf eine Datenbank erfolgt via `window.indexedDB.open()`
- ▶️ Eine Webseite kann mehr als eine Datenbank erstellen
- Jeder Zugriff ist asynchron und muss mit entsprechenden Events behandelt werden (z.B. `success` oder `error`)
- Daten (Objekte) liegen in einem Objektspeicher ("Tabellen")
- Für komplexere Zugriffe auf die Daten muss im Objektspeicher ein Index erstellt werden (z.B. Suche innerhalb des Inhalts bzw. Value)
- Mit einem Cursor kann über alle Daten iteriert werden

https://developer.mozilla.org/de/docs/IndexedDB/IndexedDB_verwenden
HTML5-Beispiel demo_indexeddb_01.html

Datenbank öffnen und Speicher erstellen

```
var db;

var request = window.indexedDB.open("movieDB", 1);
request.onerror = function (event) {
   console.log("Ups: " + event.target.error.message);
}
request.onsuccess = function (event) {
   db = request.result;
};

request.onupgradeneeded = function (event) {
   console.log("Upgrade von " + event.oldVersion + " auf " + event.newVersion);
   db = event.target.result;
   if (event.newVersion == 1)
   {
       var objectStore = db.createObjectStore("movies", { keyPath: "id",
                                                           autoIncrement: true });
   }
};
```

https://developer.mozilla.org/de/docs/IndexedDB/IndexedDB_verwenden
HTML5-Beispiel demo_indexeddb_01.html

Einträge hinzufügen

```
var db;
var request = window.indexedDB.open("movieDB", 1);
request.onsuccess = function (event) {
   db = request.result;

   var transaction1 = db.transaction(["movies"], "readwrite");
   transaction1.oncomplete = function (event) {
       console.log("gespeichert");
   };

   var objectStore1 = transaction1.objectStore("movies");
   var newmovie = {
       title: "Guardians of the Galaxy Vol. 2",
       year: 2017,
       cast: ['Chris Pratt', 'Zoe Saldana', 'Dave Bautista'],
       url: 'http://marvel.com/movies/movie/221/guardians_of_the_galaxy_vol_2'
   };
   var addrequest1 = objectStore1.add(newmovie);
   addrequest1.onsuccess = function (event) { console.log("hinzugefügt") };
};
```

https://developer.mozilla.org/de/docs/IndexedDB/IndexedDB_verwenden
HTML5-Beispiel demo_indexeddb_01.html

Einträge lesen

```
request.onsuccess = function (event) {
   db = request.result;
   var transaction2 = db.transaction(["movies"], "readonly");
   var objectStore2 = transaction2.objectStore("movies");
   var getrequest1 = objectStore2.get(1);
   getrequest1.onsuccess = function (event)
   {
      console.log(getrequest1.result);
   };
   // Oder via cursor alle movies holen
   var objectStore3 = transaction2.objectStore("movies");
   objectStore3.openCursor().onsuccess = function(event) {
       var cursor = event.target.result;
       if (cursor) {
           console.log(cursor.value);
           cursor.continue();
       }
   };
};
```

https://developer.mozilla.org/de/docs/IndexedDB/IndexedDB_verwenden
HTML5-Beispiel demo_indexeddb_01.html

Index erstellen und Abfragen

```
var db;
var request = window.indexedDB.open("movieDB", 2);
request.onsuccess = function (event) {
   db = request.result;
   var transaction2 = db.transaction(["movies"], "readonly");
   var index1 = transaction2.objectStore("movies").index("cast");
   var keyrange = IDBKeyRange.only("Zoe Saldana");
   var getrequest2 = index1.get(keyrange);
   getrequest2.onsuccess = function (event)
   {
       // Nur den ersten Eintrag mit der tiefsten ID
       console.log(getrequest2.result);
   };
};
request.onupgradeneeded = function (event) {
   if (event.newVersion == 2) {
       db.deleteObjectStore("movies");
       var objectStore = db.createObjectStore("movies", { keyPath: "id",
                                                          autoIncrement: true });
       objectStore.createIndex("cast", "cast", { unique:false, multiEntry:true });
   }
};
```

Geht das nicht einfacher?

JavaScript Libraries welche die Handhabung vereinfachen

- IndexedDB with usability.
  https://github.com/jakearchibald/idb
- Dexie.js - A Minimalistic Wrapper for IndexedDB
  http://dexie.org/
- pouchdb - The Database that Syncs!
  https://pouchdb.com/
- Lovefield - a relational database for web apps
  https://google.github.io/lovefield/

Storage (4 Punkte)

Kann mit der Web Storage API oder der IndexedDB gelöst werden.

- Beim Login (Index-Seite) soll der User-Nickname und die User-ID lokal im Browser gespeichert werden. Die User-ID wird beim Erstellen eines Users automatisch von der Chat-API zurückgegeben, die ID kann also erst nach der AJAX Übung (siehe unten) gespeichert werden.
- Beim erneuten öffnen der Login-Seite soll der gespeicherte User-Nickname bereits im Login-Nickname-Feld stehen.
- Alle Felder auf der Profilseite sollen beim Absenden des Formulars ebenfalls lokal gespeichert werden und beim erneuten Besuch der Seite entsprechend angezeigt werden (inklusive korrekte Selektion der Radio-Elemente des Avatars und des Backgrounds).

Web Sockets

Protokoll, Websockets API

WebSocket-Protokoll

Allgemeines

- Auf TCP basiertes Protokoll für bidirektionale Verbindungen zwischen Webseite und Server
- ws://url oder wss://url (mit TLS Verschlüsselung)
- Bevor die Verbindung zustande kommt, findet ein Handshake statt
  - Der Header ist dabei abwärtskompatibel zu HTTP
- Zuständig für die Standardisierung ist die IETF
- Web Sockets brauchen immer eine entsprechende Serverkomponente

Handshake

- **Anfrage**

```
GET / HTTP/1.1
Host: echo.websocket.org
Connection: Upgrade
Upgrade: websocket
Origin: http://websocket.org
Sec-WebSocket-Version: 13
User-Agent: Mozilla/5.0...
Sec-WebSocket-Key: qJEV1CUjjCDtER5f/Jy+Lw==
Sec-WebSocket-Extensions: permessage-deflate;
client_max_window_bits
```

- **Antwort**

```
HTTP/1.1 101 Web Socket Protocol Handshake
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Accept: 3ZplML+dGdhNVv162e5Z089HSpg=
```

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
https://tools.ietf.org/html/rfc6455

Websockets API

WebSocket Interface

- Methoden
  - **WebSocket**(url, [subprotocol])
  - **send**(data)
  - **close**()
- Eigenschaften
  - **binaryType**
  - **bufferedAmount**
  - **extensions**
  - **protocol**
  - **readyState**
  - **url**
- Events
  - **message**
  - **close**
  - **error**
  - **open**

Sub-Protokoll und Extensions

- Beim öffnen kann der Client ein bestimmtes Sub-Protokoll für die Datenübertragung innerhalb des Websockets anfordern
- Sub-Protokolle sind bei der IANA Registriert
- In der Regel einfach leer lassen

- Extensions erweitern die Funktionalität des WebSocket-Protokolls (z.B. Komprimierung)
- Sie werden automatisch ausgehandelt und können mit JavaScript nur abgefragt werden

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications
https://www.html5rocks.com/de/tutorials/websockets/basics/
https://www.codetinkerer.com/2018/06/05/aspnet-core-websockets.html
HTML5-Beispiel demo_websocket.html

Websockets API

Kommunikation

- Mit der send-Methode kann Text oder können Daten in binärer Form (`ArrayBuffer` oder `Blob`) übertragen werden
- Die Binäre-Form muss via `binaryType`-Eigenschaft angegeben werden (default ist "blob")
- JavaScript Objekte können serialisiert übertragen werden (`JSON.stringify` und `JSON.parse`)

- Nachrichten vom Server lösen den message-Event mit einem `MessageEvent` Objekt aus welches die Daten enthält

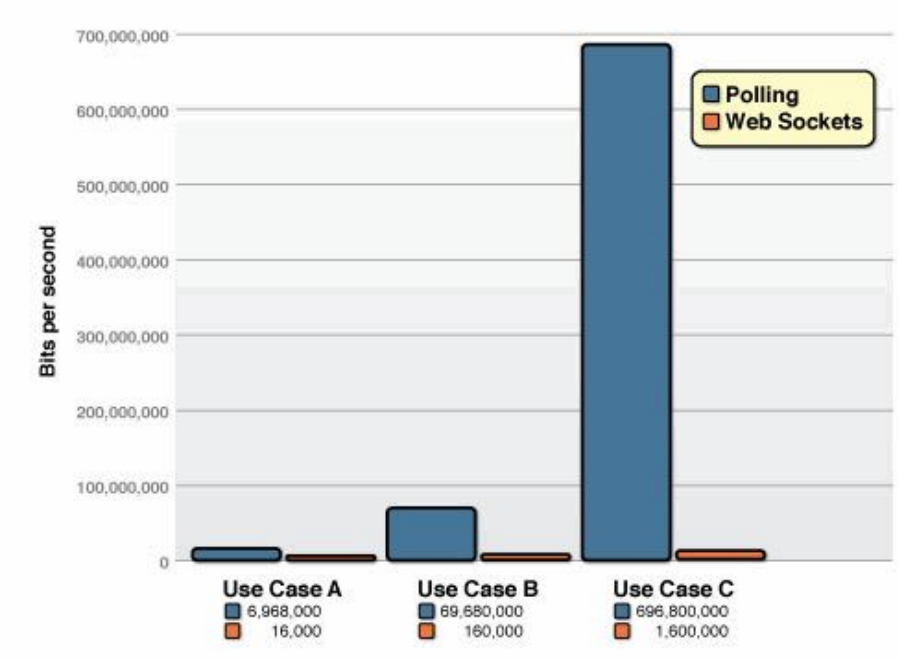
- Eine saubere Server- und Clientseitige Implementation ist nicht trivial (Authentifizierung, Unterbrüche, …)

http://www.websocket.org/quantum.html
https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events

Alternativen

So wie “früher”…

- **Polling**
  Client fragt immer wieder den Server an
- **Long-Polling**
  Der Server versucht die Anfrage des Clients so lange wie möglich offen zu halten bis eine neue Nachricht eintrifft



Oder allenfalls mit server-sent events

- Nur Senden, IE/Edge werden (noch) nicht unterstützt

https://moodle.bfh.ch/mod/assign/view.php?id=1183856

Web Socket (4 Punkte)

- Bei einer neuen Chat-Nachricht soll der entsprechende Web Socket Event ("message_added") abgefangen und die neue Nachricht dargestellt werden.
- Zusätzlich soll mindestens einer der folgenden Web Socket Events implementiert werden: "user_added", "user_updated" und/oder "user_deleted".

Drag & Drop

Drag and Drop API

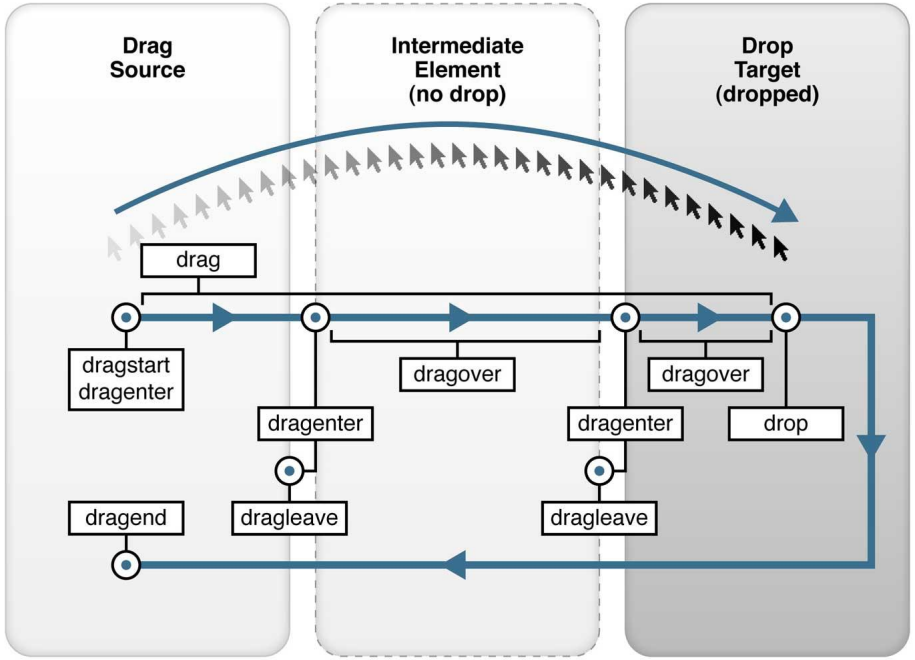
https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations
https://wiki.selfhtml.org/wiki/JavaScript/Drag_%26_Drop
http://apress.jensimmons.com/v5/pro-html5-programming/ch9.html

Drag & Drop API

Voraussetzung

- Desktop-Browser…
- Bei einem Element das **draggable**-Attribut auf `true` setzen
  - Bei Links, Images und selektiertem Text ist dies per Default gesetzt
- Einen Listener dem **dragstart**-Event anfügen
- Im Listener **Drag-Daten** hinzufügen mit `event.dataTransfer.setData()`
- Beim Zielelement müssen mindestens 2 Event-Listener definiert werden:
  - **dragover**: Soll das Ablegen möglich sein, muss mit `event.preventDefault()` das Standardverhalten (nichts machen) verhindert werden
  - **drop**: Abschliessen des Vorgangs

Events



https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations#dragdata
https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Recommended_drag_types
HTML5 Beispiele demo_dragndrop_01.html und demo_dragndrop_02.html

Drag-Daten

- Drag-Daten sind in jedem Drag & Drop Event als DataTransfer Objekt enthalten, können jedoch nur während dem dragstart-Event hinzugefügt werden.
- Ein Dateneintrag hat jeweils einen Typ und einen Inhalt (beide als String).
- Der Typ kann im Prinzip beliebig gewählt werden. Er kann z.B. zum Überprüfen verwendet werden ob ein Drop erlaubt werden soll.
- Drag & Drop funktioniert auch ohne Drag-Daten. Zum Beispiel:

```
var sourceElement
someElement.addEventListener('dragstart', (event) =>
{
  sourceElement = event.target
})
```

Empfohlene Drag-Typen

- text/plain
- text/html
- text/uri-list

MDN empfiehlt, als letzter Eintrag die Daten immer auch als text/plain mitzugeben. Dies dient als Fallback falls das Ziel den eigentlichen Typ nicht kennt.

Zum Beispiel für einen Link die URL als text/uri-list und auch als text/plain anfügen:

```
var dt = event.dataTransfer
dt.setData("text/uri-list",
           "https://www.mozilla.org")
dt.setData("text/plain",
           "https://www.mozilla.org")
```

https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer

DataTransfer Eventobjekt

Methoden

- **setData**(format, data)
  - Format der Daten z.B. “text/plain” oder “text/uri-list” oder auch “image/png”
- **getData**(format): Daten lesen
- **clearData**([format])

- **setDragImage**(img, xOffset, yOffset)
  - Bild welches während dem Vorgang angezeigt wird.

Der Zugriff auf die Daten sind je nach Event eingeschränkt, vollzugriff hat nur der dragstart-Event, der drop-Event kann die Daten einfach lesen.

Das Bild kann bei jedem Drag & Drop Event angepasst werden.

Eigenschaften

- **effectAllowed**
  - `none, copy, copyLink, copyMove, link, linkMove, move, all, uninitialized`
  - Festlegen der erlaubten Operation. Geht nur während dem dragstart Event
- **dropEffect**
  - `none, copy, link or move`
  - Aktuelle Operation, kann überschrieben werden
- **files**
  - Liste mit Dateien (`FileList`) um Drag & Drop von lokalen Dateien zu ermöglichen
- **items** und **types**
  - Liste mit Drag-Daten bzw. Typen

Drag & Drop (2 Punkte)

- Bei der Liste mit den Usern soll die Reihenfolge via Drag & Drop geändert werden können. Also ein User soll an eine andere Position verschoben werden können.

Die restlichen Seiten kommen nächste Woche…

(https://moodle.bfh.ch/mod/resource/view.php?id=1183855)