

Web-Entwicklung

Clientseitige Entwicklung II: Dynamische Web-Anwendungen

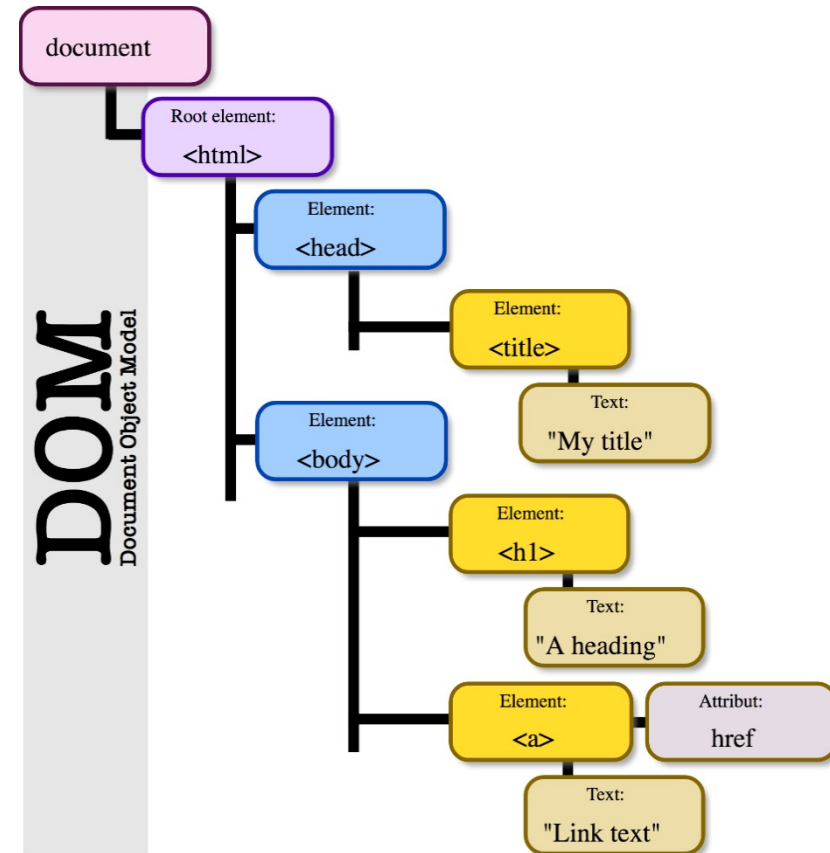
Inhalte der Vorlesung

- Clientseitige Entwicklung: Dynamische Web-Anwendungen
 - DOM-Manipulation
 - DOM-Events
 - AJAX
 - XMLHttpRequest
 - Fetch
 - WebSocket

DOM-Manipulation

Document Object Model (DOM)

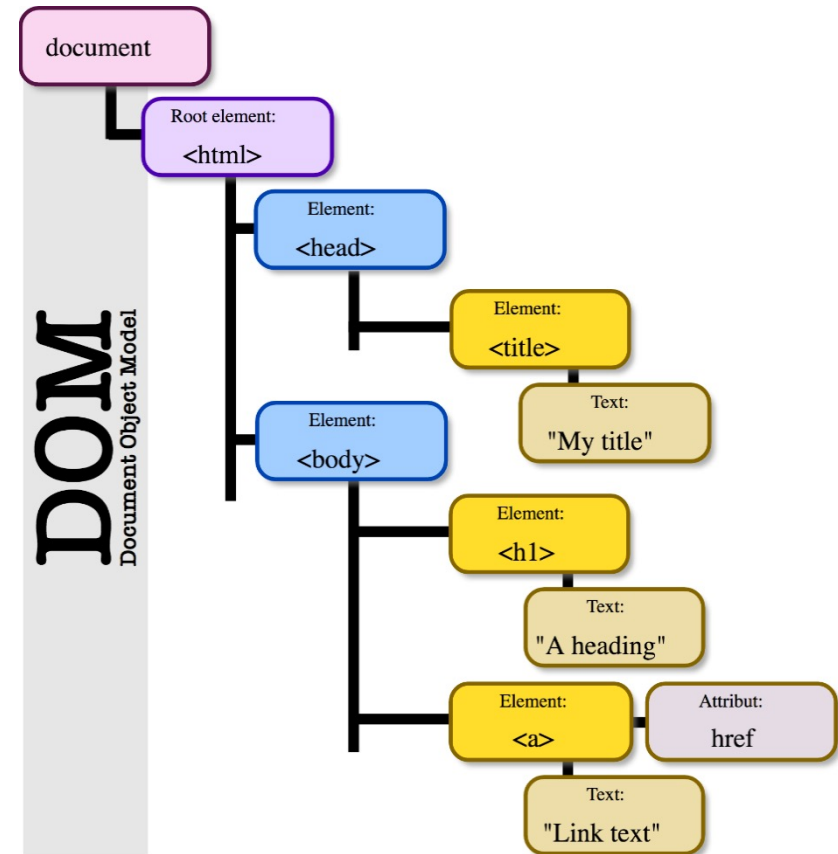
- Sammlung von Web APIs zum Zugriff auf Webseiteninhalte
 - Spezifikation durch W3C
 - Implementierungen durch die Browser-Hersteller



Quelle: Wikipedia, Birger Eriksson

Document Object Model (DOM)

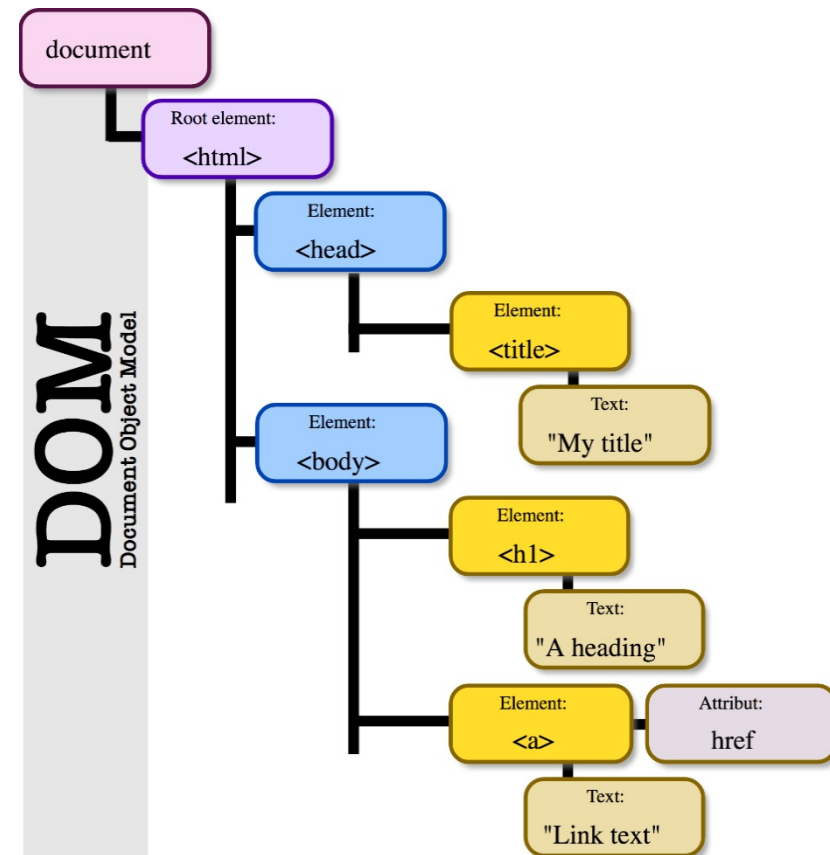
- Zugrundeliegende Datenstruktur: Baum
 - DOM-Baum
 - DOM-Knoten (implementieren Schnittstelle Node)



Quelle: Wikipedia, Birger Eriksson

Document Object Model (DOM)

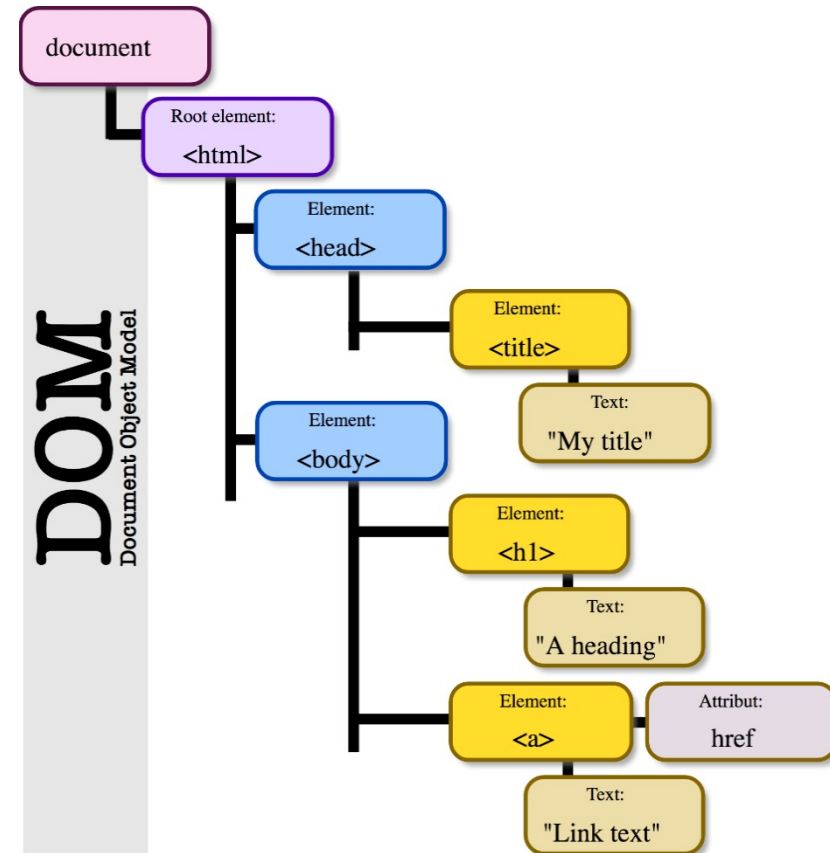
- Document mit implizit erzeugter Instanz `document`
- Elementknoten (Element)
 - z.B. `main`, `header`, `footer`, `div`, `h1`, `p`, `ul`, `li`, `a`, ...
- Attributknoten (Attr)
 - z.B. `id`, `class`, `href`, `style`, ...
- Textknoten (Text)
 - `<p>Hello, World!</p>`
 - Immer ein Blatt
- ...



Quelle: Wikipedia, Birger Eriksson

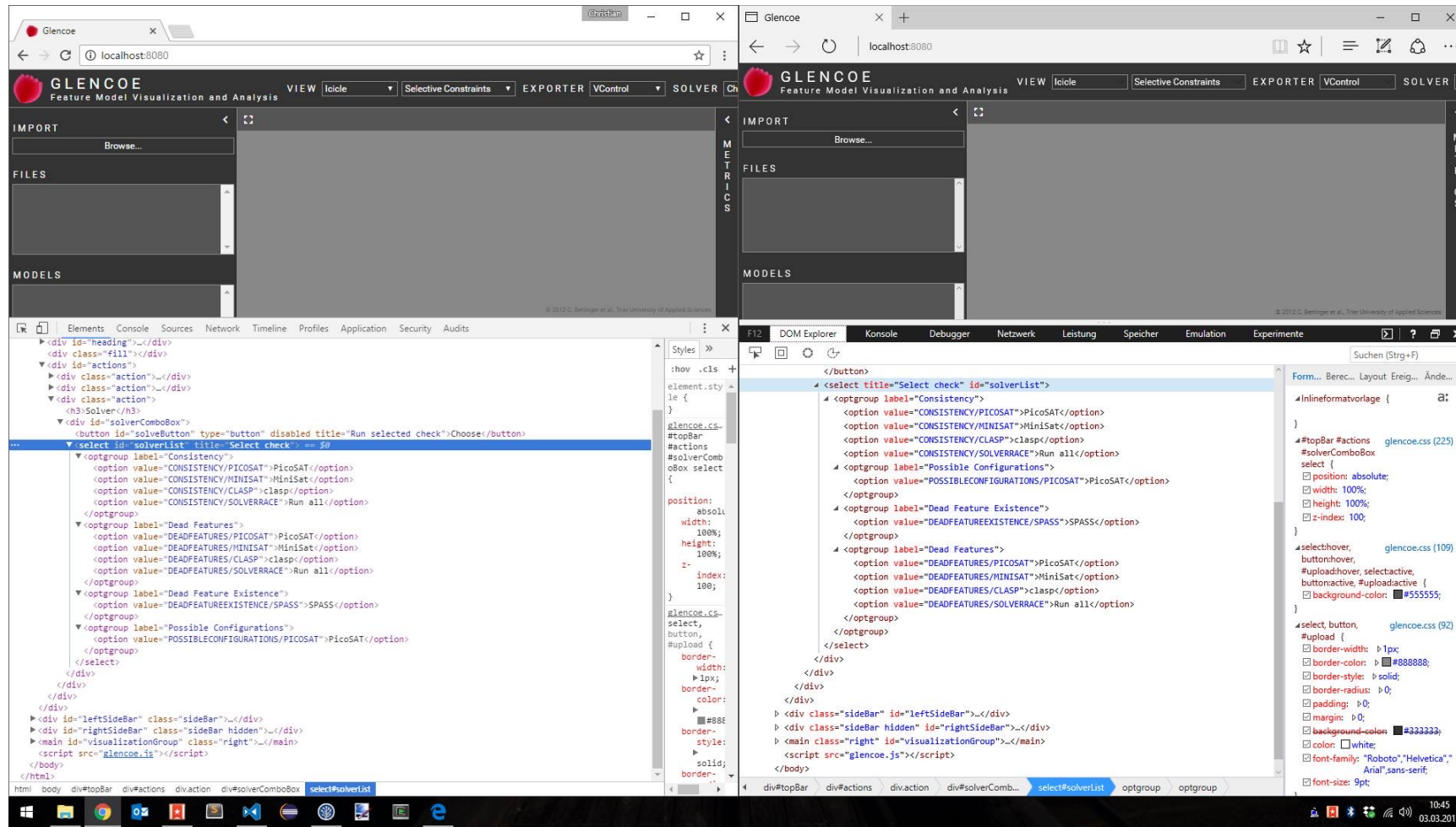
Document Object Model (DOM)

- Sammlung von Web APIs zum Zugriff auf Webseiteninhalte
 - <https://developer.mozilla.org/en-US/docs/Web/API/Node>
 - <https://developer.mozilla.org/en-US/docs/Web/API/Element>
 - <https://developer.mozilla.org/en-US/docs/Web/API/Document>



Quelle: Wikipedia, Birger Eriksson

Document Object Model (DOM)



Elemente selektieren

- Einzelnes Element
 - `Document.getElementById()`
 - `Document.querySelector()`
- Liste von Elementen
 - `Document.getElementsByClassName()`
 - `Document.getElementsByTagName()`
 - `Document.querySelectorAll()`
 - Bis auf `getElementById()` können alle Methoden auch auf einem Elementknoten statt auf dem Dokumentknoten aufgerufen werden und durchsuchen somit nur einen Teilbaum
- Tipps:
 - Defensiv programmieren – Existenz der Rückgabe prüfen
 - Wenn selektiertes Element mehrmals benötigt wird, Rückgabe der Methode in Variable speichern
 - Bevorzuge bei Selektion eines eindeutigen Elements `getElementById()`, bei komplexeren (d.h. mehrstufigen) Selektionsausdrücken mit einer Ergebnismenge `querySelectorAll()`

Elemente selektieren

```
<body>
  <h1>Hello World!</h1>
  <p class="important">Very important: Lorem ipsum...</p>
  <h2>Hello World!</h2>
  <p>Lorem ipsum...</p>
  <ul id="list">
    <li>Item A</li>
    <li>Item B</li>
    <li>Item C</li>
    <li>Item D</li>
    <li>Item E</li>
  </ul>
  <p class="important">Most important: Lorem ipsum...</p>
  <h2>Hello World!</h2>
  <p>Lorem ipsum...</p>
</body>
```

```
let list = document.getElementById("list");
let listItems = list.getElementsByTagName("li");
for (let li of listItems) {
  console.log(li.innerHTML);
}

let importantParagraphs = document.
  getElementsByClassName("important");
for (let p of importantParagraphs) {
  console.log(p.innerHTML);
}
```

Item A
Item B
Item C
Item D
Item E
Very important: Lorem ipsum dolor sit amet, consetetur erat, sed diam voluptua.
Most important: Lorem ipsum dolor sit amet, consetetur erat, sed diam voluptua.

Elemente selektieren

- `querySelector()` und `querySelectorAll()` liefern Elemente, die einem CSS-Selektor genügen
 - siehe Vorlesung "Web-Technologien"
 - War vor Einführung der Selector API nur mithilfe von Fremdbibliotheken (z.B. jQuery) möglich
- Vergleichsweise "teuer", dafür mächtig und komfortabel
- `querySelectorAll()` liefert alle selektierten Elemente
- `querySelector()` liefert nur das erste selektierte Element

Elemente selektieren

```
<body>
  <ul id="nestedList">
    <li>A</li>
    <ul>
      <li>A1</li>
      <li>A2</li>
      ...
    </ul>
    <li>B</li>
    <ul>
      <li>B1</li>
      <li>B2</li>
      ...
    </ul>
    ...
  </ul>
</body>
```

```
let firstListItemsInSubLists = document
    .querySelectorAll("#nestedList ul li:first-child");
for (let li of firstListItemsInSubLists) {
    console.log(li.innerHTML);
}
```

A1
B1
C1

Elemente traversieren

- Ausgehend von einem selektierten Element müssen häufig "benachbarte" Elemente betrachtet werden
- Eigenschaften der Schnittstelle `Element` traversieren nur Elemente
 - `parentElement`
 - `children`, `firstElementChild`, `lastElementChild`
 - `childElementCount`
 - `previousElementSibling`, `nextElementSibling`
- Weitere Eigenschaften der Schnittstelle `Node` traversieren alle Knoten
 - `parentNode`
 - `childNodes`, `firstChild`, `lastChild`
 - `hasChildNodes()`
 - `previousSibling`, `nextSibling`

Elemente traversieren

```
<body>
  <ul id="nestedList">
    <li>A</li>
    <ul>
      <li>A1</li>
      <li>A2</li>
      ...
    </ul>
    <li>B</li>
    <ul>
      <li>B1</li>
      <li id="b2">B2</li>
      <li>B3</li>
    </ul>
    ...
  </ul>
</body>
```

```
let b2 = document.getElementById("b2");
let path = "";
let current = b2;
while (current) {
  path = path.length === 0 ? `${current.tagName}` :
    `${current.tagName} > ${path}`;
  current = current.parentElement;
}
console.log(path);

let siblings = b2.parentElement.children;
for (let sibling of siblings) {
  if (sibling !== b2) { console.log(sibling.innerHTML); }
}
```

HTML > BODY > UL > UL > LI

B1

B3

Elemente ändern

- Textinhalt: `Node.textContent`
 - Streng genommen: Textinhalt des untergeordneten Textknotens eines Element
- Attribute
 - `Element.getAttribute()`
 - `Element.setAttribute()`
 - `Element.removeAttribute()`
- CSS-Klassenattribut `class`
 - `Element.classList.add()`
 - `Element.classList.remove()`
 - `Element.classList.toggle()`
 - `Element.classList.contains()`

Elemente erzeugen und einfügen

- Erzeugung
 - `Document.createElement()`
 - `Document.createTextNode()`
 - `Document.createAttribute()` (→ `setAttribute()`)
- In den DOM-Baum einfügen bzw. entfernen
 - `Node.appendChild()`
 - `Node.insertBefore()`
 - `Node.replaceChild()`
 - `Node.removeChild()`

Elemente erzeugen und einfügen

```
<body>
  ...
  <div id="timesTable"></div>
  <script src="dom.js"></script>
</body>
```



```
<body>
  ...
  <table id="timesTable">
    <tr><td>1</td><td>2</td>...<td>10</td></tr>
    <tr><td>2</td><td>4</td>...<td>20</td></tr>
    ...
  </table>
  <script src="dom.js"></script>
</body>
```

```
let table = document.createElement("table");
table.setAttribute("id", "timesTable");

for (let rowIndex = 1; rowIndex <= 10; rowIndex++) {
  let row = document.createElement("tr");
  for (let colIndex = 1; colIndex <= 10; colIndex++) {
    let cell = document.createElement("td");
    cell.textContent = rowIndex * colIndex;
    row.appendChild(cell);
  }
  table.appendChild(row);
}

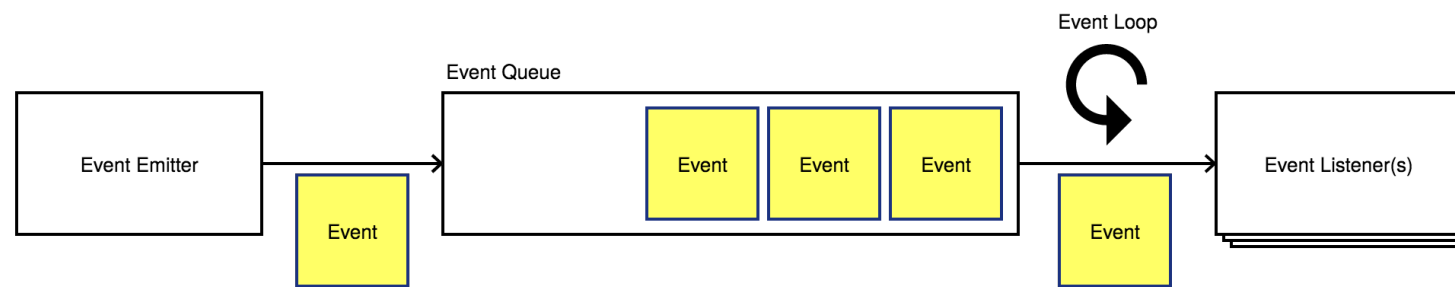
let timesTable = document
  .getElementById("timesTable");
timesTable.parentNode.replaceChild(table, timesTable);
```

DOM-Events

Events

- In allen Programmierumgebungen mit grafischer Benutzeroberfläche spielen Events eine zentrale Rolle
- Nach Ausführung des Einstiegspunktes übernehmen sie die Steuerung des Programmflusses
- Event Emitter: Ein Objekt signalisiert, dass ein bestimmtes Ereignis eingetreten ist
 - DOM-Events: Event Emitter ist immer ein DOM-Element
 - Spätere Vorlesungseinheit: Allgemeiner Event-Mechanismus
- Event Listener: Beliebige Objekte können Funktionen beim Event Emitter registrieren, die dann bei Eintritt eines bestimmten Ereignisses aufgerufen werden (Callback-Mechanismus)
 - Mehrere Listener für das gleiche Event am gleichen Objekt möglich
- Entwurfsmuster: Observer

Events



DOM-Events

- Selektion des DOM-Elements, auf dessen Events man reagieren möchte
 - Oder eines Elements im Pfad der Eltern (*Event Bubbling*)
- Registrieren eines Event Listeners an diesem Element mithilfe der Methode `addEventListener()` unter Angabe
 - des Event-Typs (String)
 - einer (anonymen) Callback-Funktion
 - die als Argument ein Objekt mit Prototyp `Event` (oder mit einem davon abgeleiteten Prototyp) erhält
- Abmelden eines Event Listeners mithilfe der Methode `removeEventListener()`
 - Im Hinblick auf die Speicherbereinigung sollten Event Listener abgemeldet werden, wann immer möglich und sinnvoll

Event-Typen

- Event
 - Eigenschaft: target
- MouseEvent
 - Typen: click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseleave
 - Eigenschaften: button, clientX, clientY, screenX, screenY, altKey, ctrlKey, shiftKey, metaKey, ...
- KeyboardEvent
 - Typen: keydown, keyup, keypress
 - Eigenschaft: key
- FocusEvent
 - Typen: focusin, focusout
- UIEvent
 - Typen: load, unload, abort, error, resize, scroll, ...
- ...

Beispiel: Registrieren eines Event Listeners

```
(function () {  
    let currentSize = 10;  
    createTable(currentSize);  
})();
```

```
function createTable(size) {  
    let table = document.createElement("table");  
    table.setAttribute("id", "timesTable");  
    for (let rowIndex = 1; rowIndex <= size; rowIndex++) {  
        let row = document.createElement("tr");  
        for (let colIndex = 1; colIndex <= size; colIndex++) {  
            let cell = document.createElement("td");  
            cell.textContent = rowIndex * colIndex;  
            row.appendChild(cell);  
        }  
        table.appendChild(row);  
    }  
    let timesTable = document.getElementById("timesTable");  
    timesTable.parentNode.replaceChild(table, timesTable);  
}
```

Beispiel: Registrieren eines Event Listeners

```
(function () {  
    let currentSize = 10;  
    createTable(currentSize);  
  
    // keyboard event listener  
    document.addEventListener("keyup", event => {  
        if (event.key === "+") {  
            currentSize++;  
        }  
        else if (event.key === "-") {  
            currentSize = Math.max(1, currentSize - 1);  
        }  
        createTable(currentSize);  
    });  
})();
```

```
function createTable(size) {  
    let table = document.createElement("table");  
    table.setAttribute("id", "timesTable");  
    for (let rowIndex = 1; rowIndex <= size; rowIndex++) {  
        let row = document.createElement("tr");  
        for (let colIndex = 1; colIndex <= size; colIndex++) {  
            let cell = document.createElement("td");  
            cell.textContent = rowIndex * colIndex;  
            row.appendChild(cell);  
        }  
        table.appendChild(row);  
    }  
    let timesTable = document.getElementById("timesTable");  
    timesTable.parentNode.replaceChild(table, timesTable);  
}
```


Beispiel: Registrieren eines Event Listeners

```
(function () {  
    let currentSize = 10;  
    createTable(currentSize);  
  
    // keyboard event listener  
    document.addEventListener("keyup", event => {  
        if (event.key === "+") {  
            currentSize++;  
        }  
        else if (event.key === "-") {  
            currentSize = Math.max(1, currentSize - 1);  
        }  
        createTable(currentSize);  
    });  
})();
```

```
function createTable(size) {  
    let table = document.createElement("table");  
    table.setAttribute("id", "timesTable");  
    for (let rowIndex = 1; rowIndex <= size; rowIndex++) {  
        let row = document.createElement("tr");  
        for (let colIndex = 1; colIndex <= size; colIndex++) {  
            let cell = document.createElement("td");  
            cell.textContent = rowIndex * colIndex;  
            row.appendChild(cell);  
            // mouse event listener (on each cell)  
            cell.addEventListener("dblclick",  
                event => console.log(event.target.textContent));  
        }  
        table.appendChild(row);  
    }  
    let timesTable = document.getElementById("timesTable");  
    timesTable.parentNode.replaceChild(table, timesTable);  
}
```

Beispiel: Registrieren eines Event Listeners

```
(function () {  
    let currentSize = 10;  
    createTable(currentSize);  
  
    // keyboard event listener  
    document.addEventListener("keyup", event => {  
        if (event.key === "+") {  
            currentSize++;  
        }  
        else if (event.key === "-") {  
            currentSize = Math.max(1, currentSize - 1);  
        }  
        createTable(currentSize);  
    });  
})();
```

```
function createTable(size) {  
    let table = document.createElement("table");  
    // mouse event listener (on complete table)  
    table.addEventListener("dblclick",  
        event => console.log(event.target.textContent));  
    table.setAttribute("id", "timesTable");  
    for (let rowIndex = 1; rowIndex <= size; rowIndex++) {  
        let row = document.createElement("tr");  
        for (let colIndex = 1; colIndex <= size; colIndex++) {  
            let cell = document.createElement("td");  
            cell.textContent = rowIndex * colIndex;  
            row.appendChild(cell);  
        }  
        table.appendChild(row);  
    }  
    let timesTable = document.getElementById("timesTable");  
    timesTable.parentNode.replaceChild(table, timesTable);  
}
```

Eigene DOM-Events

- Bisher: Event Emitter werden (i.d.R. als Reaktion auf eine Nutzeraktion) implizit durch die DOM-Elemente ausgelöst
- Jetzt: Programmatische Auslösung von DOM-Events durch den Entwickler
 - Erlaubt lose gekoppelte Kommunikation zwischen Objekten
- Prinzipieller Ablauf
 - Objekt mit Prototyp Event erstellen
 - Erstes Konstruktorargument erwartet Typ-String
 - Übergabe des Objekts an Aufruf der Methode `dispatchEvent()` an dem Event Emitter
- Falls Event Emitter Daten an den Event Listener übermitteln möchte, kann Objekt mit Prototyp `CustomEvent` erstellt werden
 - Zweites Konstruktorargument erwartet Konfigurationsobjekt mit Eigenschaft `detail`

Beispiel: Eigene DOM-Events auslösen

```
(function () {  
    document.addEventListener("keyup", event => {  
        // dispatch custom events  
        if (event.key === "t") {  
            document.dispatchEvent(new Event("myEvent"));  
            document.dispatchEvent(new CustomEvent("myCustomEvent", {  
                detail: {  
                    time: new Date().toUTCString()  
                }  
            }));  
        }  
    });  
  
    // listen for custom events  
    document.addEventListener("myEvent", event => console.log(event.type));  
    document.addEventListener("myCustomEvent", event => console.log(event.detail.time));  
})();
```

Beispiel: Eigene DOM-Events auslösen

```
(function () {  
    document.addEventListener("keyup", event => {  
        // dispatch custom events  
        if (event.key === "t") {  
            document.dispatchEvent(new MyTimeEvent());  
        }  
    });  
  
    // listen for custom events  
    document.addEventListener(MyTimeEvent.ON_TIME,  
        event => console.log(event.getTime()));  
})();
```

```
class MyTimeEvent extends CustomEvent {  
    constructor() {  
        super(MyTimeEvent.ON_TIME, {  
            detail: {  
                time: new Date().toUTCString()  
            }  
        });  
    }  
  
    getTime() {  
        return this.detail.time;  
    }  
}  
  
MyTimeEvent.ON_TIME = "MYTIMEEVENT_ON_TIME";
```

AJAX

Frontend - Backend

- Kommunikation der clientseitigen Browser-Anwendung ("Frontend") mit entfernten Servern ("Backends") ist ein wesentlicher Aspekt von Web-Anwendungen
 - Eines der wesentlichen Unterscheidungskriterien zwischen Web-Anwendungen und einfachen Websites
- Typisches Szenario: Zugriff auf einen zentralen Dienst
 - Dynamische Generierung von Website-Inhalten aus zentraler Datenquelle
 - Single Page Applications
 - Verlagerung von Verarbeitungsschritten (z.B. Berechnungen) auf einen Server
 - Tendenz zu dezentraler Verarbeitung

Frontend - Backend

- Wichtigste Kommunikationsprotokolle
 - HTTP
 - WebSocket
- Protokoll dient als Schnittstelle und erlaubt Realisierung des Backends unabhängig vom Frontend mithilfe verschiedener Technologien
- Diese Vorlesungseinheit: Clientseitige Realisierung
 - Serverseitige Realisierung in nächster Vorlesungseinheit

AJAX

- Asynchronous JavaScript and XML
 - Zeitweilig war auch synchrone Nutzung möglich
 - Inzwischen *deprecated*
 - Freie Wahl des Austauschformats
 - In den vergangenen Jahren starker Trend zu JSON
- Jesse James Garrett: A New Approach to Web Applications (2005)
 - Seinerzeit fundamental neues Konzept zur Interaktion zwischen Browser und serverseitiger Anwendung
 - Einer der wichtigsten Meilensteine auf dem Weg zu modernen Web-Anwendungen

AJAX

- Ohne AJAX
 - Klicken eines Links oder Absenden eines Formulars führt dazu, dass der Browser eine HTTP-Anfrage an den Server sendet
 - Server generiert Antwortseite und sendet diese zurück
 - Browser ersetzt die alte Seite durch die neue Antwortseite
- Probleme
 - Synchrone Kommunikation
 - Zu übertragende Datenmenge relativ groß
 - Kein direkter Einfluss auf den HTTP-Client
 - Problem für REST-basierte Anwendungen

AJAX – Single Page Applications

- Mit AJAX: Single Page Applications
 - Einmaliger Download der gesamten Seite beim initialen Aufruf
 - Browser stellt der clientseitigen JavaScript-Anwendung einen parametrisierbaren HTTP-Client zur Verfügung
 - Web API XMLHttpRequest
 - Realisiert in allen gängigen Browsern
 - Anschließende Aktionen, die eine HTTP-Kommunikation mit dem Server bedingen, werden asynchron ausgeführt
 - Antwort des Servers ist i.d.R. keine vollständige Seite, sondern nur ein angefragter Datensatz
 - Datensatz wird via DOM-Manipulation in die Seite eingepflegt statt diese vollständig auszutauschen

AJAX

- Beispiele
 - Nachladen von Datensätzen
 - Paginierung von großen Datensätzen
 - Auto-Vervollständigung von Formularfeldern
 - Editierbare UI-Elemente
 - ...

XMLHttpRequest: GET

```
(function () {  
  let request = new XMLHttpRequest();  
  request.addEventListener("error", error => { console.error(error.toString()); });  
  request.addEventListener("load", () => { if (request.status === 200) { listFilms(request.response); } });  
  request.open("GET", "https://swapi.co/api/films/");  
  request.setRequestHeader("Accept", "application/json");  
  request.responseType = "json";  
  request.send();  
  
  function listFilms(films) {  
    films = films.results.sort((a, b) => a.episode_id - b.episode_id);  
    let list = document.getElementById("films");  
    for (let i = 0; i < films.length; i++) {  
      let item = document.createElement("li");  
      item.textContent = films[i].title;  
      list.appendChild(item);  
    }  
  }  
})());
```

Austauschformate

- Andere gängige Austauschformate
 - XML
 - HTML (falls Server bereits gerenderte HTML-Fragmente sendet)
- Gelegentlich auch proprietäre oder binäre Formate
- Viele APIs können Daten im mehreren Formaten ausliefern
 - Angabe des gewünschten Formats über Festlegung des MIME-Types, z.B.
`request.setRequestHeader("Accept", "application/json")`
`request.setRequestHeader("Accept", "text/xml")`
`request.setRequestHeader("Accept", "text/html")`
 - Wert in `request.response` legt Prototyp des Objekts in `request.responseType` fest, z.B.
`request.responseType = "text"`
`request.responseType = "json"`
`request.responseType = "document"` (für XML und HTML)
`request.responseType = "blob"` bzw.
`request.responseType = "arraybuffer"` (für Binärdaten)

XMLHttpRequest: GET

```
(function () {  
  let request = new XMLHttpRequest();  
  request.addEventListener("error", error => { console.error(error.toString()); });  
  request.addEventListener("load", () => { if (request.status === 200) { showCountryInfo(request.response); } });  
  request.open("GET", "http://api.geonames.org/countryInfo?lang=de&country=LU&username=cbettinger");  
  request.setRequestHeader("Accept", "text/xml");  
  request.responseType = "document";  
  request.send();  
  
  function showCountryInfo(info) {  
    let country = info.getElementsByTagName("country")[0];  
    let table = document.getElementById("countryInfo");  
  
    for (let child of country.children) {  
      let row = document.createElement("tr");  
      let keyColumn = document.createElement("td"); keyColumn.textContent = child.tagName; row.appendChild(keyColumn);  
      let valueColumn = document.createElement("td"); valueColumn.textContent = child.textContent; row.appendChild(valueColumn);  
      table.appendChild(row);  
    }  
  }  
})();
```

Daten senden

- XMLHttpRequest unterstützt neben GET auch die anderen HTTP-Verben: POST, PUT, DELETE, OPTIONS, HEAD, ...
 - Wichtig für REST-basierte Anwendungen
 - Siehe nächste Vorlesungseinheit
- Gängige Szenarien
 - Senden von beliebigen Daten über eine POST-Anfrage
 - Senden von Formulareingaben

XMLHttpRequest: POST

```
(function () {  
    let data = {  
        episode_id: 8,  
        title: "A Stupid Idea"  
    };  
  
    let request = new XMLHttpRequest();  
    request.addEventListener("error", error => { console.error(error.toString()); });  
    request.addEventListener("load", () => { if (request.status === 200) { console.log("Success"); } });  
    request.open("POST", "https://swapi.co/api/films/");  
  
    request.setRequestHeader("Content-Type", "application/json");  
    request.send(data);  
})();
```

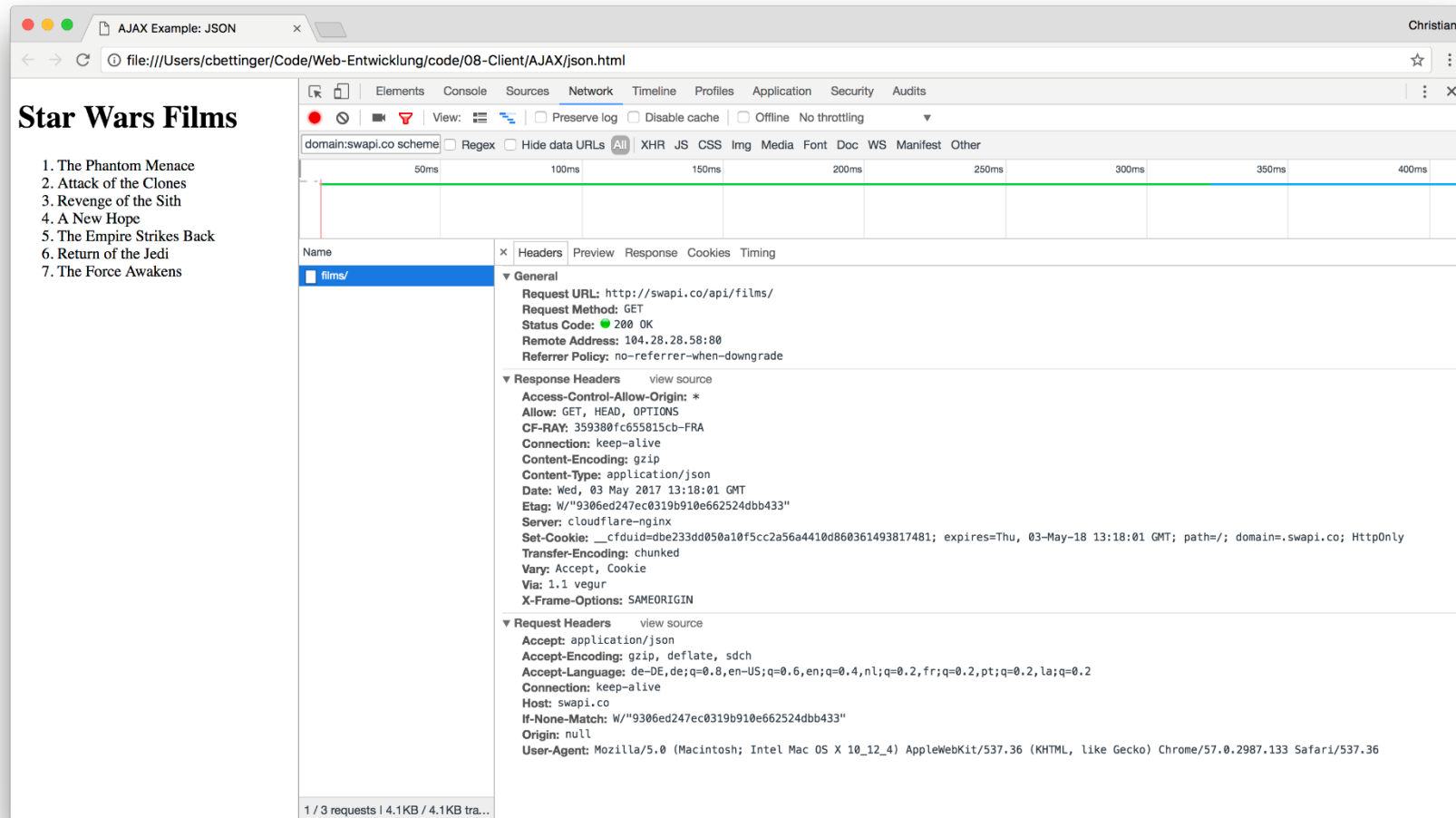
XMLHttpRequest: FormData

```
(function () {  
    let form = document.getElementById("addFilmForm");  
    form.addEventListener("submit", event => {  
        event.preventDefault();  
        let formData = new FormData(form);  
        let request = new XMLHttpRequest();  
        request.addEventListener("error", error => { console.error(error.toString()); });  
        request.addEventListener("load", () => { if (request.status === 200) { console.log("Success"); } });  
        request.open("POST", "https://swapi.co/api/films/");  
        request.setRequestHeader("Content-Type", "application/json");  
        request.send(formData);  
    }  
})();
```

Same-Origin-Policy

- Sicherheitskonzept: Browser verweigern JavaScript-Anwendungen den Zugriff auf Ressourcen mit anderem Ursprung als dem der Website, welche die Anwendung geladen hat
 - Explizit: Protokoll und Host (inkl. Portnummer) müssen übereinstimmen
`request.open("GET", "https://glencoe.hochschule-trier.de/list");`
 - Implizit: Relative URL
`request.open("GET", "/list");`
- Die vorangegangenen Beispiele dürften demnach nicht funktionieren!
- Öffentliche APIs sollen aber von anderen Web-Anwendungen konsumiert werden können.
- Lösungsansätze
 - Proxy
 - Cross-Origin-Resource-Sharing (CORS)
 - Express-Middleware: <https://github.com/expressjs/cors>

Cross-Origin-Resource-Sharing (CORS)



Fetch

- Moderne Web API als Ersatz für XMLHttpRequest
- Globale Funktion `fetch()` liefert Promise-Objekt
 - 1. Argument: URL
 - 2. optionales Argument: Konfigurationsobjekt (HTTP-Verb, Header, Body, ...)
- Im Erfolgsfall kann über den Aufruf von entsprechenden Methoden erneut ein Promise-Objekt erstellt werden, das die asynchrone Antwort des Servers über einen Datenstrom ausliest und in ein Objekt umwandelt
 - `text()`
 - `json()`
 - `formData()`
 - `blob()`
 - `arrayBuffer()`
- Kann (wie jedes Promise-Objekt) auch mit `async/await` genutzt werden

Fetch: GET

```
(function () {  
    fetch("https://swapi.co/api/films/").then(response => {  
        if (response.ok) { return response.json(); }  
    }).then(result => {  
        listFilms(result);  
    }).catch(error => {  
        console.error(error.message);  
    });  
})();
```

```
(async function () {  
    try {  
        let response = await fetch("https://swapi.co/api/films/");  
        let result = await response.json();  
        listFilms(result);  
    }  
    catch (error) {  
        console.error(error.message);  
    }  
})();
```

Fetch: POST

```
(function () {  
    fetch(https://swapi.co/api/films/, {  
        method: "POST",  
        headers: {  
            "Content-Type": "application/json"  
        },  
        body: JSON.stringify({ episode_id: 8, title: "A Bad Idea" })  
    }).then(response => {  
        if (response.ok) { return response.json(); }  
    }).then(result => {  
        console.dir(result);  
    }).catch(error => {  
        console.error(error.message);  
    });  
})();
```

WebSocket

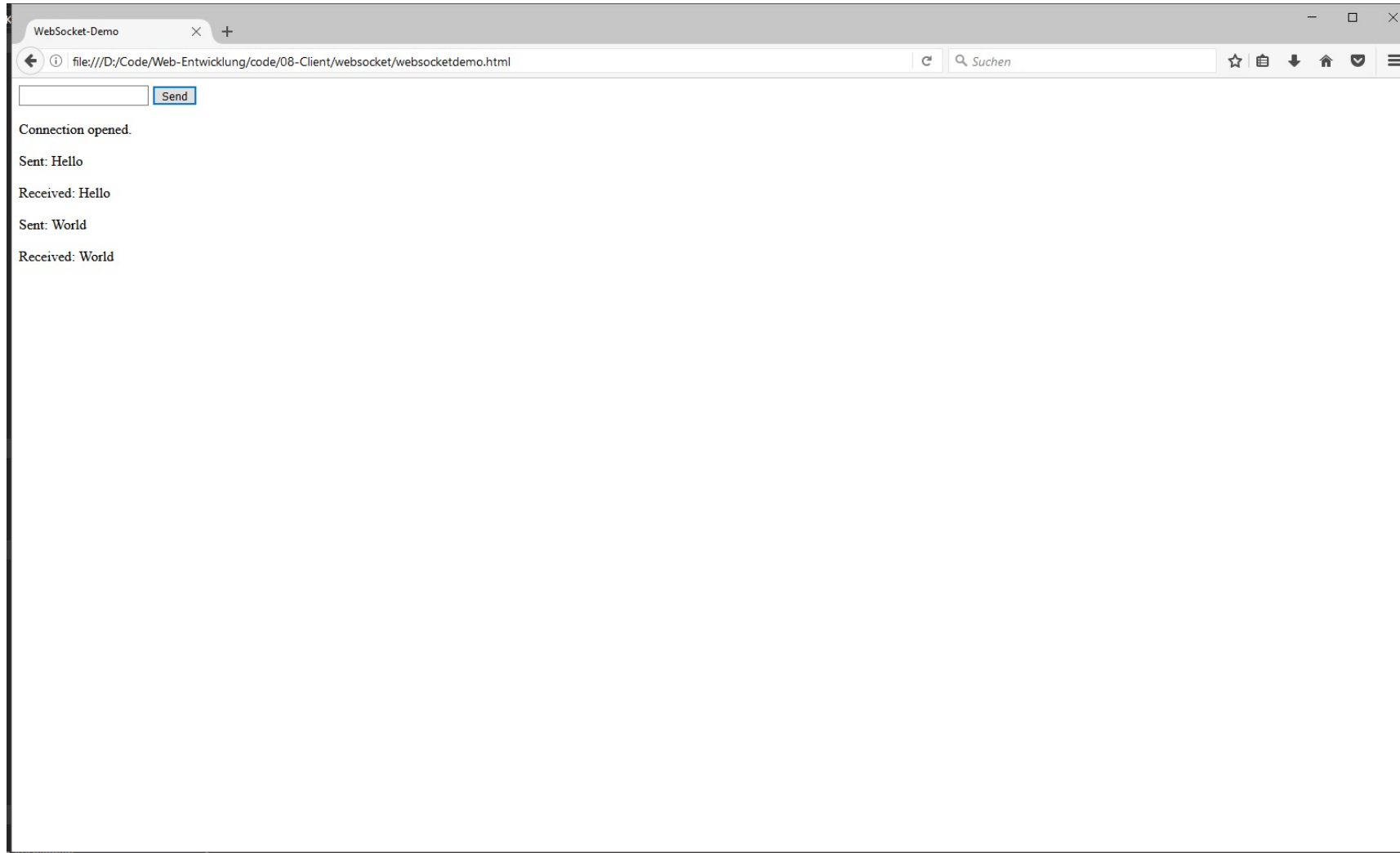
WebSocket

- Kommunikation via HTTP hat auch Nachteile, z.B.
 - Alle Anfragen müssen vom Client initiiert werden (Pull), Server kann ohne explizite Anfrage keine Daten an den Client übertragen (Push)
 - Datenmenge im HTTP-Header des Browsers in der Praxis häufig recht hoch
 - Wird in jeder HTTP-Anfrage mitgesendet
 - Zugrundeliegende TCP-Verbindung wird nach jeder Anfrage geschlossen
 - Latenz
- Alternative: WebSocket-Protokoll
 - Schicht 5-Protokoll, aufbauend auf TCP (wie HTTP)
 - Etabliert eine ständige bidirektionale Verbindung
 - Handshake ist HTTP-abwärtskompatibel und benötigt daher keinen zusätzlichen Port

WebSocket-Client

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>WebSocket-Demo</title>
  <script defer src="websocketclient.js"></script>
</head>
<body>
  <input type="text" id="inputField" />
  <button type="button" id="sendButton">Send</button>
  <main></main>
</body>
</html>
```

WebSocket-Client



WebSocket-Client

```
let output = document.getElementsByTagName("main")[0];

function addMessage(message, prefix = "") {
  if (output && message) {
    let line = document.createElement("p");
    line.textContent = prefix + message;
    output.appendChild(line);
  }
}
```

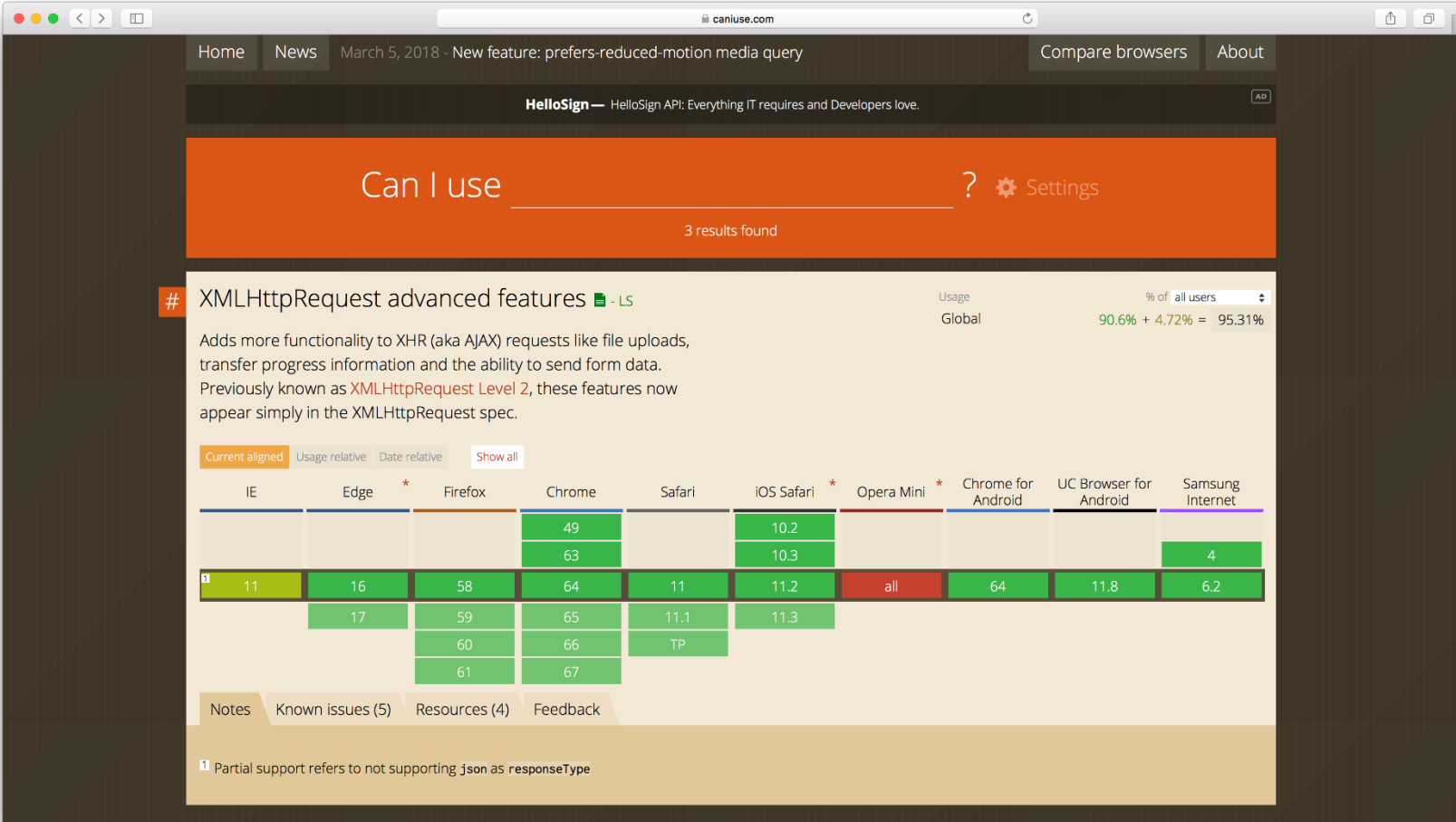
WebSocket-Client

```
(function () {  
    let connection = new WebSocket("ws://localhost:8080");  
    connection.onopen = event => { addMessage("Connection opened."); };  
    connection.onerror = event => { addMessage("Unable to connect."); };  
    connection.onclose = event => { addMessage("Connection closed."); };  
    connection.onmessage = event => { addMessage(event.data, "Received: "); };  
  
    let inputField = document.getElementById("inputField");  
    let sendButton = document.getElementById("sendButton");  
    sendButton.addEventListener("click", () => {  
        if (inputField.value) {  
            addMessage(inputField.value, "Sent: ");  
            connection.send(inputField.value);  
            inputField.value = "";  
        }  
    });  
})();
```

Welche API einsetzen?

- Pull: Unidirektionale aktive Kommunikation (vom Client ausgehend) ist ausreichend
 - Projekt setzt vorrangig Callbacks ein → `XMLHttpRequest`
 - Projekt setzt vorrangig Promises ein → `fetch()`
- Push: Bidirektionale aktive Kommunikation (nach Verbindungsaufbau durch den Client) wird benötigt
 - `WebSocket`
 - `WebSocket` + `XMLHttpRequest/fetch()`
 - Falls neue Daten auf dem Server vorhanden sind, pusht dieser einen simplen "Ping" per `WebSocket` an jene Clients, für die diese Information relevant sein könnte
 - Clients pullen – durch den "Ping" getriggert – Daten via `XMLHttpRequest/fetch()`

Browser-Unterstützung: XMLHttpRequest



The screenshot shows the 'Can I use' section for 'XMLHttpRequest advanced features' on the caniuse.com website. The page includes a search bar, a list of features, and a table showing browser support percentages.

Can I use ? [Settings](#)

3 results found

XMLHttpRequest advanced features [- LS](#)

Usage: Global 90.6% + 4.72% = 95.31%

Adds more functionality to XHR (aka AJAX) requests like file uploads, transfer progress information and the ability to send form data. Previously known as **XMLHttpRequest Level 2**, these features now appear simply in the XMLHttpRequest spec.

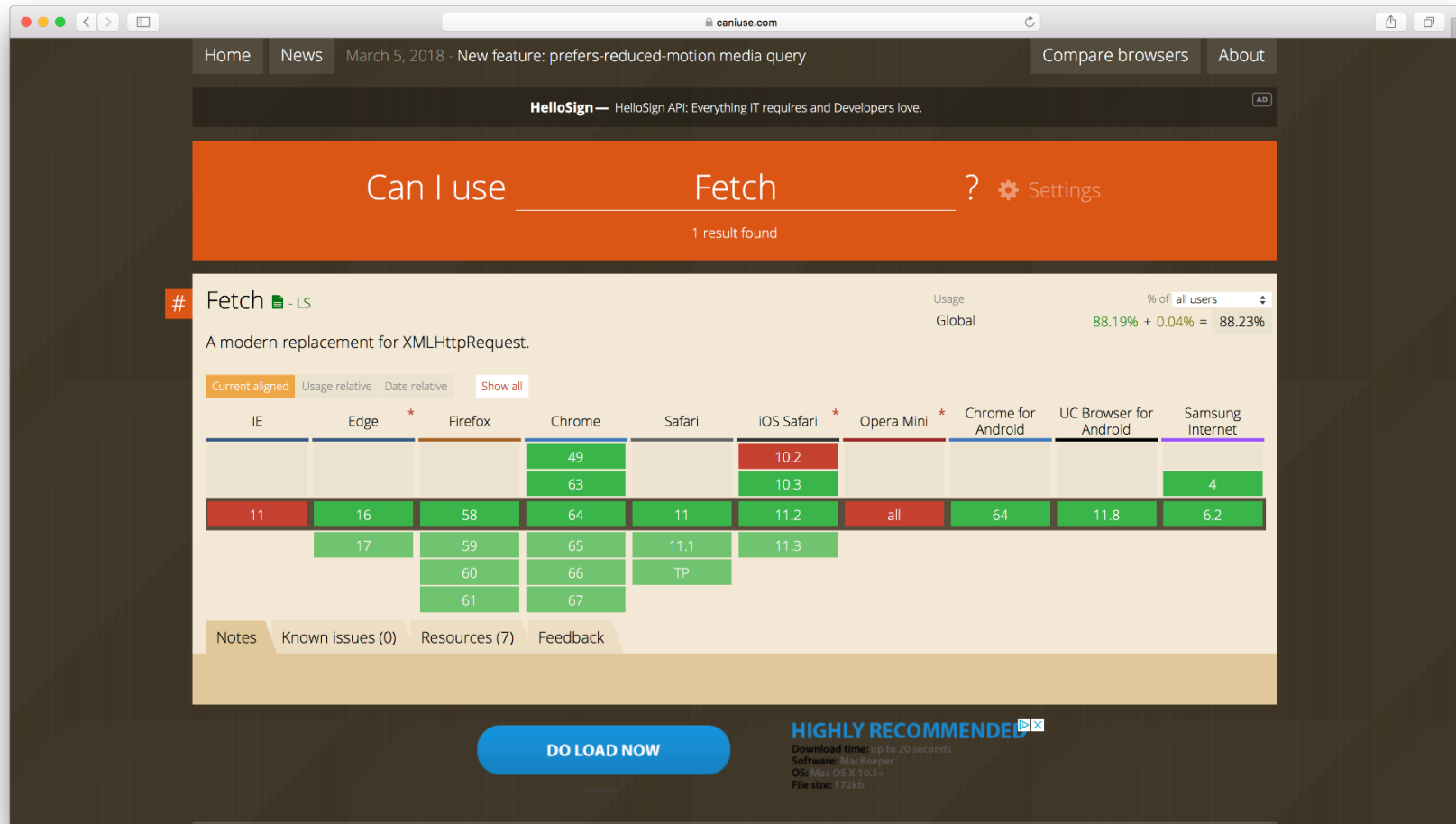
[Current aligned](#) [Usage relative](#) [Date relative](#) [Show all](#)

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Opera Mini	Chrome for Android	UC Browser for Android	Samsung Internet
			49		10.2				
			63		10.3				4
11	16	58	64	11	11.2	all	64	11.8	6.2
	17	59	65	11.1	11.3				
		60	66	TP					
		61	67						

[Notes](#) [Known issues \(5\)](#) [Resources \(4\)](#) [Feedback](#)

1 Partial support refers to not supporting json as responseType

Browser-Unterstützung: Fetch



Browser-Unterstützung: WebSocket

caniuse.com

Home News March 5, 2018 - New feature: prefers-reduced-motion media query Compare browsers About

HelloSign — HelloSign API: Everything IT requires and Developers love.

Can I use Web Sockets ? Settings

1 result found

Web Sockets LS

Bidirectional communication technology for web apps

Usage % of all users

Global 94.77% + 0.27% = 95.04%

unprefixed: 94.77% + 0.23% = 95%

Current aligned Usage relative Date relative Show all

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49		10.2				
			63		10.3				4
11	16	58	64	11	11.2	all	64	11.8	6.2
	17	59	65	11.1	11.3				
		60	66	TP					
		61	67						

Notes Known issues (1) Resources (10) Feedback

Reported to be supported in some Android 4.x browsers, including Sony Xperia S, Sony TX and HTC.

DO LOAD NOW

HIGHLY RECOMMENDED

Download time: up to 20 seconds
Software: MacKeeper
OS: Mac OS X 10.5+
File size: 172kb

Fragen?

© 2015 Christian Bettinger

Nur zur Verwendung im Rahmen des Studiums an der Hochschule Trier.

Diese Präsentation einschließlich aller Abbildungen ist urheberrechtlich geschützt. Jede Verwertung außerhalb der Grenzen des Urheberrechts ist ohne Zustimmung des Autors unzulässig.

Die Quellen der Abbildungen sind entsprechend angegeben. Alle Marken sind das Eigentum ihrer jeweiligen Inhaber, wobei alle Rechte vorbehalten sind.

Die Haftung für sachliche Fehler ist ausgeschlossen.