

WBE: BROWSER-TECHNOLOGIEN

CLIENT-SERVER-INTERAKTION

ÜBERSICHT

- Formulare
- Cookies, Sessions
- Ajax und XMLHttpRequest
- Fetch API

ÜBERSICHT

- Formulare
- Cookies, Sessions
- Ajax und XMLHttpRequest
- Fetch API

WARUM FORMULARE?

- Ermöglichen Benutzereingaben
- Grundlage für Interaktion im Web

```
<form>  
  <input type="text" id="name">  
  <input type="text" id="age">  
  <input type="submit" value="Send">  
</form>
```

FORMULARE: form

```
<form action="/login" method="post">
...
</form>
```

- Attribut `action`: Script das die Daten entgegen nimmt
- Attribut `method`: HTTP-Methode zum Senden der Daten
 - `GET`: Formulardaten an URL angehängt (Default)
 - `POST`: Formulardaten im Body des HTTP-Request gesendet

Beispiel: GET-Request

<http://www.google.com/search?client=safari&q=http>

BESCHREIBUNG: label

```
1 <form>
2   <label>Name: <input type="text"></label>
3   <label>Age: <input type="text"></label>
4   <input type="submit" value="Send">
5 </form>
6
7 <form>
8   <label for="nameid">Name: </label>
9   <input type="text" id="nameid">
10  <label for="ageid">Age: </label>
11  <input type="text" id="ageid">
12  <input type="submit" value="Send">
13 </form>
```

Name: Age:

- Das label-Element *beschreibt* ein Formularelement
- Klick auf Label setzt den Fokus auf das Element

FORMULARELEMENTE

```
1 <form>
2   <label>Text field <input type="text" value="hi"></label>
3   <label>Password <input type="password" value="hi"></label>
4   <label>Textarea <textarea>hi</textarea></label>
5   <label>Checkbox <input type="checkbox"></label>
6   <label>Radio button <input type="radio" name="demo" checked></label>
7   <label>Another one <input type="radio" name="demo"></label>
8   <label>Button <button>Click me</button></label>
9   <label>Select box
10    <select name="cars">
11      <option value="volvo">Volvo</option>
12      <option value="saab">Saab</option>
13      <option value="fiat">Fiat</option>
14      <option value="audi">Audi</option>
15    </select>
16  </label>
17  <input type="submit" value="Send">
18 </form>
```

Text field	<input type="text" value="hi"/>
Password	<input type="password" value=".."/>
Textarea	<input type="text" value="hi"/>
Checkbox	<input type="checkbox"/>
Radio button	<input checked="" type="radio"/>
Another one	<input type="radio"/>
Button	<input type="button" value="Click me"/>
Select menu	<input type="text" value="Volvo"/>
	<input type="button" value="Send"/>

GRUPPIEREN VON FORMULARELEMENTEN

```
1 <form>
2   <fieldset>
3     <legend>General information</legend>
4     <label>Text field <input type="text" value="hi"></label>
5     <label>Password <input type="password" value="hi"></label>
6     <label class="area">Textarea <textarea>hi</textarea></label>
7   </fieldset>
8   <fieldset>
9     <legend>Additional information</legend>
10    <label>Checkbox <input type="checkbox"></label>
11    <label>Radio button <input type="radio" name="demo" checked></label>
12    <label>Another one <input type="radio" name="demo"></label>
13  </fieldset>
14  <label>Button <button>Click me</button></label>
15  <label>Select menu
16    <select name="cars">
17      <option value="volvo">Volvo</option>
18      <option value="saab">Saab</option>
19      <option value="fiat">Fiat</option>
20      <option value="audi">Audi</option>
21    </select>
22  </label>
23  <input type="submit" value="Send">
24 </form>
```

DAS **type**-ATTRIBUT VON **input**

- Spezielle Tastaturlayouts für Mobilgeräte möglich
- Werte: `email`, `url`, `range`, `date`, `search`, `color`, ...



FOKUS

- Aktives Element
- Bei Textfeldern: das welches Tastatureingaben aufnimmt
- Referenz in `document.activeElement`
- Anpassbar mit Methoden `focus` und `blur`

```
<input type="text">
<script>
  document.querySelector("input").focus()
  console.log(document.activeElement.tagName) // → INPUT
  document.querySelector("input").blur()
  console.log(document.activeElement.tagName) // → BODY
</script>
```

INAKTIVE ELEMENTE

- Attribut `disabled` (braucht keinen Wert)
- Element ist nicht verwendbar und nicht fokussierbar
- Darstellung als inaktiv

```
<button>I'm all right</button>
<button disabled>I'm out</button>
```

I'm all right I'm out

FORMULAR-EVENTS

- Formularelement geändert:
`change`
- Eingabe in Textfeld:
`input`
- Tastendrucke in aktivem Textfeld:
`keydown`, `keypress`, `keyup`
- Formular absenden:
`submit`

FILE-INPUT: DATEI WÄHLEN

- Ursprünglich: Upload einer Datei zum Server
- Mittlerweile auch per JavaScript zugänglich
- Sicherheit: Datei muss aktiv ausgewählt werden

```
<input type="file">
<script>
  let input = document.querySelector("input")
  input.addEventListener("change", () => {
    if (input.files.length > 0) {
      let file = input.files[0]
      console.log("You chose ", file.name)
      if (file.type) console.log("It has type ", file.type)
    }
  })
</script>
```

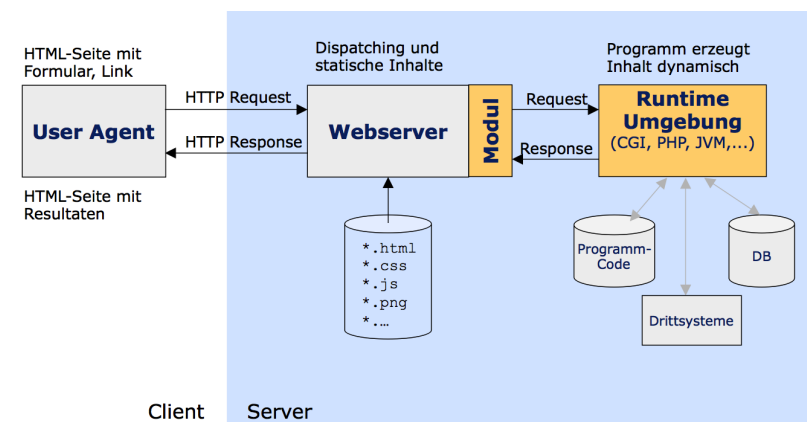
FILE-INPUT: DATEI LESEN

```
<input type="file" multiple>
<script>
  let input = document.querySelector("input")
  input.addEventListener("change", () => {
    for (let file of Array.from(input.files)) {
      let reader = new FileReader()
      reader.addEventListener("load", () => {
        console.log("File", file.name, "starts with",
          reader.result.slice(0, 20))
      })
      reader.readAsText(file)
    }
  })
</script>
```

FORMULARDATEN ÜBERTRAGEN

- Daten via HTTP an den Server übertragen
- Erforderlich: serverseitige Programmlogik
- Alternative: clientseitige Verarbeitung von Formulardaten
- Absenden mit Submit-Button `type="submit"`
- Löst ein `submit`-Event aus
- Im Event Handler kann das Absenden verhindert werden (`preventDefault`)

SERVERSEITIGE PROGRAMMLOGIK



GET-METHODE

```
<form action="http://localhost/cgi/showenv.cgi"
method="get">
<fieldset>
<legend>Login mit GET</legend>
<label for="login_get">Benutzername:</label>
<input type="text" id="login_get" name="login"/>
<label for="password_get">Passwort:</label>
<input type="password" id="password_get" name="password"/>
<label for="submit_get"></label>
<input type="submit" id="submit_get" name="submit" value="Anmelden" />
</fieldset>
</form>
```

– Login mit GET

Benutzername:	<input type="text" value="testuser"/>
Passwort:	<input type="password" value="*****"/>
<input type="button" value="Anmelden"/>	

```
GET /cgi/showenv.cgi?login=testuser&password=12345%3F&submit=Anmelden HTTP/1.1
Host localhost
User-Agent Mozilla/5.0 (Macintosh; ...) Gecko/20100101 Firefox/31.0
Accept text/html,application/xhtml+xml;q=0.9,*/*;q=0.8
Accept-Language de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding gzip, deflate
Connection keep-alive
```

GET-METHODE

```
GET /cgi/showenv.cgi?login=testuser&password=12345%3F&submit=Anmelden HTTP/1.1
```

- Formulardaten an URL angehängt (Browser-Adressleiste)
- Genannt: **Query String** (Teil der URL)
- Vor allem für passive Anfragen (zum Beispiel eine Suche)
- Ungeeignet für Anmeldedaten und andere sensiblen Informationen
- Im Beispiel: `%3F` steht für `?` (URL encoding)
- Funktionen: `encodeURIComponent`, `decodeURIComponent`

POST-METHODE

- Im Formular wird `method="post"` gesetzt
- Formulardaten im Body des HTTP-Request gesendet
- Um Daten anzulegen oder zu aktualisieren
- Für sensitive Daten (Login) kein GET verwenden

```
POST /~bkrt/cgi/showenv.cgi HTTP/1.1
Host localhost
...
Content-Type application/x-www-form-urlencoded
Content-Length 44

login=testuser&password=1234%3F&submit=Anmelden
```

GET: EXPRESS

```
// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse
console.dir(req.query.order)
// => 'desc'

console.dir(req.query.shoe.color)
// => 'blue'

// route: /user/:name
// GET /user/tj
console.dir(req.params.name)
// => 'tj'
```

POST: EXPRESS

```
var express = require('express')
var app = express()

// for parsing application/json
app.use(express.json())
// for parsing application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }))

app.post('/profile', function (req, res, next) {
  console.log(req.body)
  res.json(req.body)
})
```

ÜBERSICHT

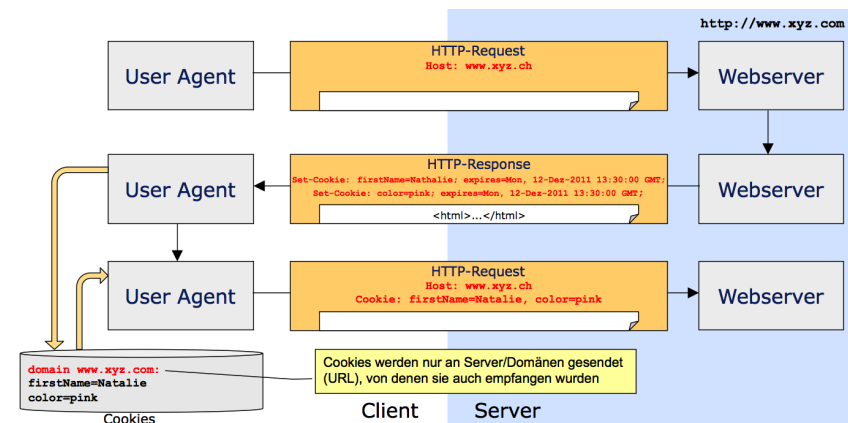
- Formulare
- Cookies, Sessions
- Ajax und XMLHttpRequest
- Fetch API

COOKIES

- HTTP als zustandsloses Protokoll konzipiert
- Cookies: Speichern von Informationen auf dem Client
- RFC 2965: HTTP State Management Mechanism
- Response: `Set-Cookie`-Header, Request: `Cookie`-Header
- Zugriff mit JavaScript möglich (ausser `HttpOnly` ist gesetzt)

```
allCookies = document.cookie
```

COOKIES

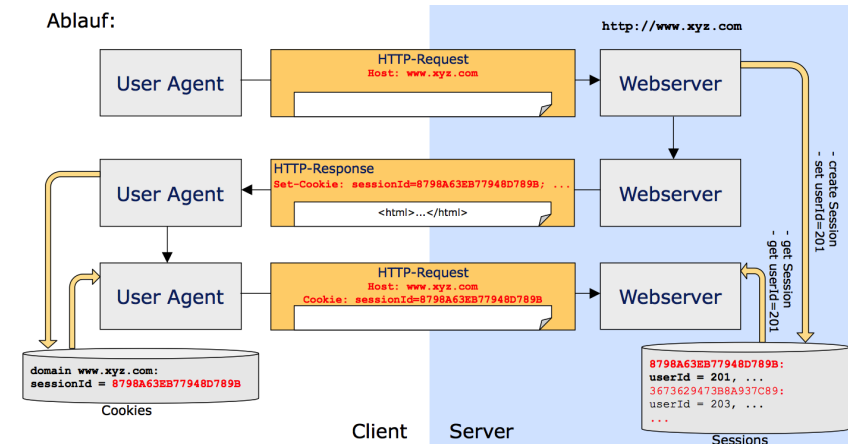


SESSIONS

- Cookies auf dem Client leicht manipulierbar
- Session: Client-spezifische Daten auf dem Server speichern
- Identifikation des Clients über **Session-ID** (Cookie o.a.)
- Gefahr: Session-ID gerät in falsche Hände (Session-Hijacking)

SESSIONS

Ablauf:



SESSIONS: EXPRESS

```
$ npm install express-session
```

```
var express = require('express')
var cookieParser = require('cookie-parser')
var session = require('express-session')
var app = express();
app.use(cookieParser())
app.use(session({secret: "Shh, its a secret!"}))

app.get('/', function(req, res){
  if(req.session.page_views){
    req.session.page_views++
    res.send("You visited this page " + req.session.page_views + " times")
  } else {
    req.session.page_views = 1
    res.send("Welcome to this page for the first time!")
  }
})
app.listen(3000)
```

ÜBERSICHT

- Formulare
- Cookies, Sessions
- **Ajax und XMLHttpRequest**
- Fetch API

TRADITIONELLES WEB-MODELL

- Seite für Seite via HTTP in Browser laden
- Für Netz von Dokumenten gut geeignet
- Für interaktive Applikationen weniger
- Idee: Informationen asynchron im Hintergrund laden
- Technische Grundlage: XMLHttpRequest-Objekt

BEZEICHNUNG: Ajax

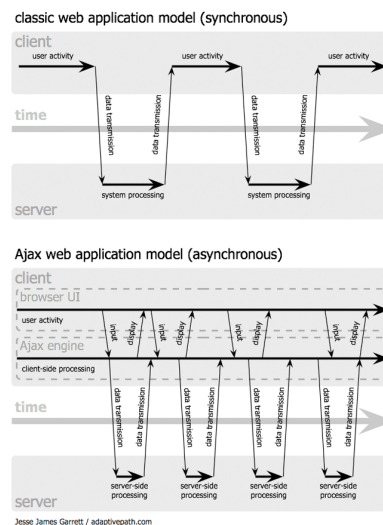
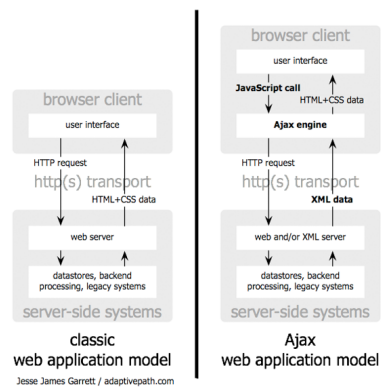
Jesse James Garrett, 2005:

Ajax: A New Approach to Web Applications

Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. It incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the DOM;
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- and JavaScript binding everything together.

AJAX



XMLHTTPREQUEST: BEISPIEL

```
const xhr = new XMLHttpRequest()
xhr.onreadystatechange = () => {
  if (xhr.readyState === 4) {
    xhr.status === 200 ? console.log(xhr.responseText)
                      : console.error('error')
  }
}
xhr.open('GET', 'https://yoursite.com')
xhr.send()
```

- Low-level, umständlich einzusetzen
- Besser Abstraktionen wie Fetch oder Axios verwenden
- XMLHttpRequest Spec:
<https://xhr.spec.whatwg.org>

ÜBERSICHT

- Formulare
- Cookies, Sessions
- Ajax und XMLHttpRequest
- Fetch API

FETCH API

- Interface for fetching resources
- Gibt Promise zurück
- Nach Server-Antwort erfüllt mit Response-Objekt

```
const url = 'https://...'  
  
fetch(url).then(response => {  
  console.log(response.status)           // → 200  
  console.log(response.headers.get("Content-Type")) // → text/plain  
})
```

RESPONSE-OBJEKT

- `headers`: Zugriff auf HTTP-Header-Daten
Methoden `get`, `keys`, `forEach`, ...
- `status`: Status-Code
- `json()`: liefert Promise mit Resultat der JSON-Verarbeitung
- `text()`: liefert Promise mit Inhalt der Server-Antwort

FETCH API

```
fetch("example/data.txt")  
  .then(resp => resp.text())  
  .then(text => console.log(text))  
// → This is the content of data.txt
```

- Promise des `fetch`-Aufrufs ist bereits nach Verarbeiten des Headers erfüllt
- Laden (und Verarbeiten) des Inhalts dauert aber länger
- Daher retournieren `json()` und `text()` selbst eine Promise
- Promise von `json()` bei Parse-Error rejected

ANDERE HTTP-METHODEN

- Normalerweise macht `fetch` eine `GET`-Anfrage
- Als zweites Argument Objekt mit weiteren Einstellungen möglich
 - `method`: HTTP-Methode
 - `headers`: HTTP Headers ergänzen
 - `body`: Inhalt mitsenden

```
fetch("example/data.txt", {method: "DELETE"}).then(resp => {  
  console.log(resp.status)  
  // → 405  
})
```

ZUM VERGLEICH: JQUERY

```
$.ajax({  
  url: "/api/getWeather",  
  data: {  
    zipcode: 97201  
  },  
  success: function (result) {  
    $("#weather-temp").html("<strong>" + result + "</strong> degrees")  
  }  
})
```

ALTERNATIVE ZU FETCH: AXIOS

- HTTP-Client für Browser und Node.js
- Ebenfalls basierend auf Promises

```
async function getUser() {  
  try {  
    const response = await axios.get('/user?ID=12345');  
    console.log(response);  
  } catch (error) {  
    console.error(error);  
  }  
}
```

<https://www.npmjs.com/package/axios>

SICHERHEIT

- Browser blockieren normalerweise Zugriffe von Scripts auf andere Domains
- Genannt: **same-origin Policy**
- Grund: Angriffsmöglichkeiten reduzieren
- Manchmal ist der Zugriff auf andere Server erwünscht
- **Cross-Origin Resource Sharing (CORS)**
- Server kann Zugriff erlauben mit diesem Header:

```
Access-Control-Allow-Origin: *
```

HTTPS

- Verschlüsselte Übertragung mittels SSL/TLS
- Unerlässlich für alle Arten sensibler Daten
- Der Standard-Port ist 443
- Server stellt Zertifikat zur Verfügung
- Browser muss dieses überprüfen können

https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure

QUELLEN

- Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
<https://eloquentjavascript.net/>
- Ältere Slides aus WEB2 und WEB3

LESESTOFF

Geeignet zur Ergänzung und Vertiefung

- Kapitel 18 von:
Marijn Haverbeke: Eloquent JavaScript, 3rd Edition
<https://eloquentjavascript.net/>
- HTML Forms (w3schools)
http://www.w3schools.com/html/html_forms.asp
- Building Forms (Shay Howe, Learn to Code HTML & CSS)
<http://learn.shayhowe.com/html-css/building-forms/>
- A Form Of Madness (Mark Pilgrim, Dive Into HTML5)
<http://diveinto.html5doctor.com/forms.html>