# html

HTML (HyperTextMarkupLanguage) beschreibt die Struktur eines Dokuments und trennt Inhalt von Darstellung.

**Diagram

Description automatically generated**

**Kategorien von Elementen:**

|  |  |
| --- | --- |
| Kategorie | Beschreibung |
| Metadaten | Beschreibt Verhalten und Darstellung des Dokuments |
| Heading | Elemente die Titel eines Abschnitts im Dokument beschreiben |
| Flow | Die meisten Elemente im <body> sind Flow content |
| Phrasing | Elemente, die Text innerhalb eines Absatzes beschreiben |

Diagram

Description automatically generated

**DOCTYPE:** DOCTYPE ist kein HTML Tag. Es ist die Info für den Browser, damit dieser weiss was für einen Dokumententyp zu erwarten ist.

<!DOCTYPE html>

**Encoding:** Browser weiss nicht wie Byte-Stream zu interpretieren. Daher soll man im Header die Encoding Informationen festlegen.

<head>  
 <meta charset="UTF-8">  
 <title>Encoding Beispiel</title>  
</head>

**Syntax/Semantik:** Syntax ist die Regel nach denen Texte strukturiert werden. Semantik ist die Zuordnung von Bedeutung zu Texten.

**Semantic Markup:** Bezieht sich auf die Kennzeichnung von Dokumenten in einer Weise, die Informationen über den Inhalt selbst liefert und nicht über die visuelle Gestaltung des Inhalts.

A picture containing logo

Description automatically generated

Diagram

Description automatically generated

**Seitenstruktur mit semantischen Elementen:**

Graphical user interface, text, application

Description automatically generated

# css

Formale Sprache um Erscheinungsbild von Dokumenten festzulegen. Wird stetig weiter entwickelt (aktuell CSS 3).

Graphical user interface, text, application, chat or text message

Description automatically generated

## selektoren

**ID:** Selektiert alle Elemente mit angegebener ID. Maximal eine ID pro Element

**Class:** Styling ähnlicher Elemente. Mehrere Klassen pro Element möglich

|  |  |
| --- | --- |
| Alle a-Elemente | a |
| Alle a-Elemente mit id important | a#important |
| Alle a-Elemente mit den Klassen news und recent | a.news.recent |

**Attribut:** Selektiert nur Elemente, die ein bestimmtes Attribut besitzen

a[href] {  
 color: rebeccapurple; }

**Pseudo-Element:** Styling von bestimmten Teilen von selektierten Elementen. Zum Beispiel ::after, ::before, ::selection etc.

*/\* The first line of every <p> element. \*/*p::first-line {  
 color: blue;  
 text-transform: uppercase;}

**Pseudo-Klassen:** Reaktion auf Zustände von selektierten Elementen. Zum Beispiel :hover, :first-child, nth-child, :visited, :active etc.

Table

Description automatically generated

**Kombinatoren:** Zwischen Selektoren kann ein Kombinator eingefügt werden. Mit :is() kann man mehrere Elemente auf einmal auswählen.

Table

Description automatically generated

**Important:** ignoriert specificity und überschreibt Wert definitiv (don’t use!)

a[href] {  
 color: rebeccapurple !important; }

Des weiteren haben hat "inherit" und "initial" specificity 0, das heisst sie werden von ALLEM überschrieben!

## CSS Werte

|  |  |
| --- | --- |
| Einheit | Relative Werte -> für fixe Werte einfach px -> in der ganzen Welt nutzbar |
| rem | basierend auf der Font-size von dem Root Element |
| em | basierend auf der Font-size vom Parent |
| vh | Basierend auf 1% Höhe des Viewports |
| vw | Basierend auf 1% Breite des Viewports |
| % | Basierend auf der Grösse des Parent |
| ch | Basierend auf der Grösse des Charakters "0" |

## sonstiges

**Flexbox:** Flexbox ist eine eindimensionale Layout-Methode für die Anordnung von Elementen in Zeilen oder Spalten. Elemente lassen sich erweitern, um zusätzlichen Platz zu füllen, und verkleinern, um in kleinere Bereiche zu passen.

<div style="display: flex"></div>

Flex-Container hat 2 Achsen, welche man mit flex-direction definieren kann. Mit row kann man die Elemte der Main-Axis der X-Achse entlang abbilden. Mit column die Y-Achse.

**Konflikte:** Falls mehrere Deklarationen ein Element betreffen wird der Konflikt nach dem spezifischten gelöst. Niedrigste Priorität sind Default-Werte des Browser gefolgt von Stylesheet von Webseite, Externe Stylesheet (</div>), interne Stylesheets (<style>) und Browser-Einstellungen.

Table

Description automatically generated**Spezifität:** Regeln werden nach Spezifität gewichtet.

**a++** = Inline-Styles

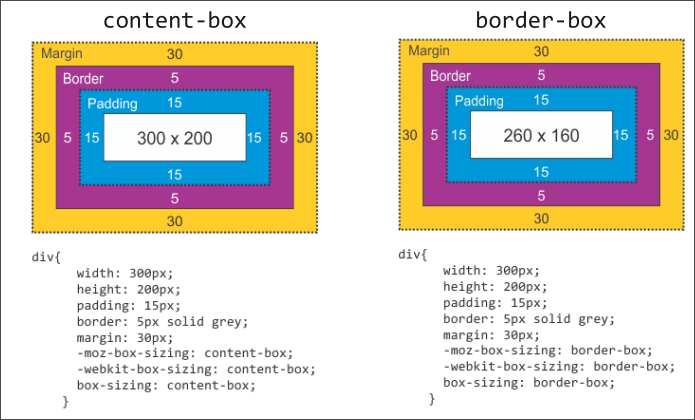
**b++** = ID-Selektoren

**c++** = Klassen-Selektoren, Attribute Pseudoklassen

**d++** = Typ-Selektoren, Pseudoelemente

Universalselektor (\*) und Pseudoklasse :not() wird ignoriert. Falls Styles gleichen Selektor haben, wird die zuletzt definierte Regel übernommen. !important überschreiben andere Deklarationen

**Boxmodell:**



**Margin/padding order:** Top, Right, Bottom, Left

# javaScript

**ECMAScript vs JavaScript:** JavaScript ist eine Implementation von ECMAScript. Generell wird JavaScript für die Programmiersprache verwendet. ECMAScript ist der Name für die Sprachdefinition.

**Dynamisch typisiert:** JavaScript ist eine dynamisch typisierte Sprache. Das heisst Variablen benötigen keine Typendeklaration und die gleiche Variable kann über die Zeit unterschiedliche Typen beinhalten. typeof() kann verwendet werden um den Typ der Variable abzufragen.

**Var:** Var sollte man nicht verwenden, da Var aufs globale Objekt gelegt wird, nicht Block-Scoped sind und man auf Var-Variablen zugreifen kann, bevor diese definiert sind.

console.log(typeof NaN); *// Number*console.log("10" + "a" \* 3); *// 10Nan*console.log((typeof [])[0]); *// "o"*let x = {  
 '1': "1",  
};  
console.log(x[1]); *// 1*x.x = x;  
x.x.test = "test";  
console.log(x.test); *// undefined*

console.log("1" + 1 \* 4); *// 14*console.log(typeof []); *// object*let arr = [];  
console.log(arr[1]); *// undefined*

console.log(NaN != NaN); *// true*console.log(null == undefined); *// true*

**Scope:** Jede Funktion und jedes Objekt generiert einen neuen Scope. Innerhalb von einem Scope kann man auf dessen Variablen, globale Variablen, Variablen aller «Parent»-Scopes zugreifen. Dieses Feature nennt sich «Closure». Diese Werte bleiben erhalten.

Text

Description automatically generated

**Beispiel:**

const funcA = function () {  
 const a = 1;  
 const funcB = function () {  
 const b = 2;  
 console.log(a, b); *// outputs: 1, 2* };  
 funcB();  
 console.log(a, b); *// Error! b is not defined*};  
funcA();  
funcB(); *// Error! funcB is not defined*

**Scope Problematik:** Global Namespace Pollution. Zu viele Variablen und Funktionen im globalen Scope. Variablen oder Funktionen mit identischem Namen überschreiben sich. Lösung dazu sind Module

**Closure:** Eine Closure ermöglicht den Zugriff auf den Bereich einer äusseren Funktion von einer inneren Funktion aus. In JavaScript werden Closures immer dann erstellt, wenn eine Funktion erstellt wird, also zur Zeit der Funktionserstellung.

function counter() {  
 let count = 10;  
 function incr() { *// incr is the inner function, a closure // has access to parent scope and keep the scope alive count++; return count;* }  
 return {incr};  
}  
const obj = counter();  
console.log(obj.count, obj.incr()); *// undefined, 11*

**Context:** «this» ist der aktuelle Context. «this» referenziert je nach Aufrufart ein anders Objekt:

* Falls ein eine Funktion als **Methode** von einem Objekt aufgerufen wird. Ist this = objekt Beispiel: object.foo();
* Falls eine Funktion mit **new()** aufgerufen wird, wird «this» mit einem neu erstellten Objekt abgefüllt. Beispiel. new Foo();
* Falls eine **«unbound» Funktion** aufgerufen wird. Zeigt «this» auf das globale Objekt.

Jeder Funktion kann mit apply() oder call() den Context gesetzt werden. In diesem Fall werden die oben genannten Regeln ignoriert.

foo.call( { counter : 123} ); bzw.

foo.apply( { counter : 123} );

Jeder Funktion kann mit bind einen Context vorgeben werden. In diesem Fall werden die oben genannten Regeln ignoriert. Diese Regel wird ignoriert falls die «gebundene Funktion» mit new aufgerufen wird. Bind() erzeugt eine neue Funktion.

const boundFoo = foo.bind({counter : 11});   
boundFoo();

**Use Strict:** Eliminiert «fails silently». Falls eine Variable ohne «var» definiert wird, falls Read-Only Werte gesetzt werden (Ohne Strict Mode: Wird einfach ignoriert). Eliminiert «Probleme» welchen es Compiler verunmöglicht den Code zu optimieren. Security wird «leicht» verbessert. Z.B. wird bei «unbound» Funktion «this» nicht auf das globale Objekt gelegt.

**Null/Undefined:** Undefined ist null in anderen Sprachen. Null in JS ist ein Wert

**Falsy/Truthy:**

|  |  |
| --- | --- |
| Falsy | Truthy |
| 0 | «0» |
| «» | «false» |
| null | [] |
| undefined | {} |
| NaN |  |

## primitives

**Boolean:** Jeder Wert kann in ein boolean gewandelt werden. 0, «», null, undefined und NaN werden zu false konvertiert während «0», «string», [], {} zu true gewandelt werden.

**Number:** Nach Definition sind alle Zahlen floats. Die Engines versuchen floats auf integers abzubilden falls möglich. NaN ist ein Error Wert, welcher eine number ist. Jeder Wert kann in eine Zahl verwandelt werden.

**String:** Text. Mit Template Literals kann ein String Placeholder besitzen.

const name = "Michael", hobby = "Hike", a = 4, b = 5;  
console.log(`Mein Name ist: ${name} Hobby: ${hobby}`);  
console.log(`${a} + ${b} = ${a + b}`);

**Equality (==):** Beschreibt, wie ein Gleichheitsvergleich von zwei Werten abläuft. Besonders aufgrund der dynamischen Typen in JavaScript. Resultiert in true oder false.

**Equality (===):** Verhindert die Typenumwandlung von Primitives.Für Objekte nicht notwendig

**Null / Undefined:** Null ist ein Wert einer Variable. Undefined ist wenn eine Variable nicht definiert oder initialisiert wurde. Z.b let a; Kann für partielle Updates verwendet werden.

Table

Description automatically generated

**Typeof:**

|  |  |
| --- | --- |
| **typeof(type)** | **Resultat** |
| Undefined | 'undefined' |
| Null | 'object' |
| Boolean | 'boolean' |
| Number | 'number' |
| BigInt (EcmaScript 2020) | 'bigint |
| String value | string' |
| Function | 'function' |
| Symbol (ECMAScript 6) | 'symbol' |
| All other | 'object' |

## array

Keine fixe Länge und beginnt bei Index 0.

**Basics:**

const arr = [ 'a', 'b', 'c' ];   
arr [0] = 'x';  
arr.push("d");  
console.log(arr); *// [ 'x', 'b', 'c', 'd' ]*console.log(arr.length); *// 4*

**Iteration for:**

*//«Klassische» for-Schleife*for(let i=0; i<arr.length; ++i) {   
 console.log("for",arr[i]);}   
*//For-In iteriert über die Property Namen*for(const x in arr) {   
 console.log("for in", x + ":" + arr[x]);}   
*//For-Of iteriert über die Werte*for(const y of arr) {   
 console.log("for of", y);}

**Iteration foreach:**

arr.forEach((elem, index) => elem);

*//For-Of iteriert über die Werte*

**Array Funktionen:**

const sum = arr.reduce((total, n) => total + n, 0);   
*// sum of all elements in array, last value is 0 (2nd param)*const mapped = arr.map((x) => x \* 2);   
*// all values doubled*const some = arr.some((x) => x === 0);   
*// true if one value in the array has value 0*const every = arr.every((x) => x === 0);   
*// true if all values are 0*const filter = arr.filter ((x) => x === 0);   
*// filter is a new array with all NON 0 values removed*const sort = arr.sort((a,b) => { return a - b});  
*// note the a - b -> in js needs to return number !! not bool!*

## object

Ein Object ist eine Sammlung von Properties, welche in einem Set verwaltet wird. Da Objekte dynamisch sind, können Properties/Methoden hinzugefügt/verändert werden. Jedes Objekt ist eine HashTable (Ausnahme null und undefined)

const person = {  
 name: "Michael",  
 hallo: function () {  
 return "Hallo " + this.name;}  
};  
person.name = "Bob";  
person.hallo();

## funktionen

Funktionen sind First-Class Citizens, d.h können in Variablen gespeichert werden, als Parameter übergeben werden und von Funktionen zurückgegeben werden. Besitzen eine offene Parameter-Liste. Es können mehr oder weniger als die deklarierte Anzahl an

Parameter übergeben werden.

function *helloWorld*(a) {  
 console.log(a || "No Data"); }

Funktionen besitzen eine offene Parameter-Liste. Arguments beinhaltet alle Parameter, welche der Funktion übergeben wurden und ist kein Array.

**Rest-Parameters:** Mit ...name kann man den letzten Parameter als Rest-Parameter definieren. Dieser Parameter wird mit allen restlichen Werten abgefüllt.

function *foo*(name, ...params){   
 console.log(1,name); console.log(2,params.join(";"));}  
*foo*("Michael", "Gfeller", "OST", "IFS");

**Properties:** Funktionen beinhalten Properties. Mit .name beinhaltet den Namen der Funktion. .length beinhaltet die Anzahl Parameter der Funktion.

console.log(fn2.name);   
console.log(fn2.length);

**Overloading:** JavaScript kennt kein Fuction Overloading. Bei gleichen Funktionsnamen überschreibt die zuletzt definierte die vorhergehende. Kann mit typoeof umgangen werden

let asdf = function (selector, context) {  
 if (!selector) {  
 return this;  
 }  
 *// Handle HTML strings* if (typeof selector === "string") {  
 *//...* } else if (jQuery.isFunction(selector)) {  
 *//...* }  
 return jQuery.makeArray(selector, this);  
};

**Default Parameter:**

function something(name = "henlo") {  
 console.log(name);  
}  
something(); *// henlo*something("birb"); *// birb*

## klassen

Class-Keyword seit ES6 vorhanden. Ähnlich wie in Java da Klassen Methoden / Properties, Vererbung, static methods / static properties und private (selten gebraucht) haben.

class Clock { *// define class* #timer *// private field* currentTime *// field (optional)* constructor(currenTime = new Date()) {  
 this.currentTime = currenTime; *// create a public property this.start();* }  
 start() {*// method* this.#timer = *setTimeout*(() => {  
 this.currentTime = new Date();  
 }, 1000);  
 }  
 *// ...* get time() { *// getter* return this.currentTime;  
 }  
 set time(newTime) { *// setter* return this.currentTime = newTime;  
 }  
}

**Vererbung:**

class AlarmClock extends Clock { *// inheritance* #timer  
 constructor() {  
 super(); *// super call* }  
 registerAlarm(time, callback) { *// method // ...* }  
 set time(value) {  
 this.#clearTimer();  
 super.time = value; *// super call* }  
 #clearTimer() {  
 if (this.#timer) {  
 *clearTimeout*(this.#timer);  
 this.#timer = null;  
 }  
 }  
}  
console.log(alarmClock instanceof AlarmClock); *// true*console.log(alarmClock instanceof Clock); *// true*

## module

Verhindern von Global namespace pollution. Skript-Tags erzeugen keinen Scope, daher werden llle Funktionen in den globalen Scope platziert. Lösung dazu ist ESM.

Diagram

Description automatically generated

Ein Module kann Funktionalität und Werte anderen Modulen zu Verfügung stellen. Ein Module kann von anderen Module exportierte Funktionalität und Werte nutzen.

*// index.html, type = module -> ESM wird aktiviert*

<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <script src="index.js" type="module"></script>  
</head>

*// index.js, mit import können andere Module mittels eines relative Pfades geladen werden*import {register as alarmClock} from './libs/alarm-clock.js'  
import {register as newsFeed} from './libs/news-feed.js'  
  
alarmClock(1, "time to eat", (msg) => {  
 console.log(msg);  
});  
newsFeed(1.5, "20min", (msg) => {  
 console.log(msg);  
});  
newsFeed(1, "blick", (msg) => {  
 console.error(msg);  
});

*// alarm-clock.js, alle Files, die mit Import geladen werden, sind automatisch auch ein Modul. Mit export werden Funktionalität und Daten freigegeben*export const register = (…) => {  
 *setTimeout*(() => {  
 callback(message);  
 }, seconds \* 1000);  
}

**Syntax:** Es gibt 2 Arten von Import / Export, Named und Default.

*// Named exports*export function *f*() {}  
export const one = 1;  
export {foo, b as bar};  
*// Default exports*export default function *f*() {} *// declaration with optional name  
// Replacement for `const` (there must be exactly one value)*export default 123;  
*// Re-exporting from another module*export {foo, b as bar} from './some-module.mjs';  
export \* from './some-module.mjs';  
export \* as ns from './some-module.mjs'; *// ES2020*

*// Named imports*import {foo, bar as b} from './some-module.mjs';  
*// Namespace import*import \* as someModule from './some-module.mjs';  
*// Default import*import someModule from './some-module.mjs';  
*// Combinations:*import someModule, \* as someModule from './some-module.mjs';  
import someModule, {foo, bar as b} from './some-module.mjs';  
*// Empty import (for modules with side effects)*import './some-module.mjs';

**Infos:** Module sind immer ‘use strict’. Module werden in der richtigen Reihenfolge geladen.

## js sprachfeatures

**Spread:**

const listA = [1, 2, 3]  
const listB = [4, 5, 6]  
const listAB = [...listA, ...listB] *// concatenate arrays console.log(listAB); // [ 1, 2, 3, 4, 5, 6 ]*const objA = {firstName: "Michael"}  
const objB = {firstLast: "Gfeller"}  
const objAB = {...objA, ...objB}] *// concatenate objects – no deep clone*console.log(objAB); *// { firstName: 'Michael', firstLast: 'Gfeller' }*console.log({...objA, objB}); *// { firstName: 'Michael', objB: { firstLast: 'Gfeller' } }*function sum(a, b, c) {  
 return a + b + c;  
}  
console.log(sum(...listA)); *// 6*

**Destructuring:**

const [a, b] = [10, 20]; *// array destructuring*console.log(a, b)  
const {c, d} = {c: 10, d: 20}; *// object destructuring*console.log(c, d)  
function print({message}) { *// object destructuring* console.log(message);  
}  
print({message: "text", code: "error\_1"})  
function printWithDefaults({message = "message"} = {}) { *// object destructuring* console.log(message);  
}  
printWithDefaults({code: "error\_1"})  
printWithDefaults()

**Nullish coalescing operator:**

console.log(null ?? 'default string'); *// default string*console.log(0 ?? 42); *// 0*console.log(0 || 42); *// 42*

**Optional chaining:**

const adventurer = {  
 name: 'Alice', cat: {  
 name: 'Dinah'  
 }  
};  
const dogName = adventurer.dog?.name;  
console.log(dogName); *// undefined*console.log(adventurer.someNonExistentMethod?.()); *// undefined*

**Shim/Polyfill:** A polyfill is a shim for a browser API. Implementieren API’s die noch nicht zu Verfügung stehen. Fixen JavaScript Bugs in alten Browsern. Polyfill können wieder entfernt werden – falls dieser nicht mehr notwendig ist.

# dom

**document:** Repräsentiert Dokument(HTMLDocument). Einstiegspunkt für DOM-Baum

**window:** Fenster, in dem das Dokument dargestellt wird. Stellt globale Objekte zur Verfügung (console, document, etc.)

**getElementsBy():** Bei Änderungen des DOM ändert sich Collection. Liefert statische NodeList

**querySelector():** Syntax entspricht CSS-Selektoren

## dom-lifecycle

**DOM-Lifecycle:** Mit document.readyState der Ladezustand des Dokuments abfragen

* **Loading:** Dokument lädt noch
* **Interactive:** Dokument geladen, Ressourcen (Scripts, Bilder etc.) werden noch geladen
* **Complete:** Dokument und Ressourcen geladen

A picture containing text, clipart

Description automatically generated

**DOMContentLoaded (Interactive):** Libraries oder Skripte, die nicht mit anderen Ressourcen interagieren.

**Load (Complete):** Falls Aktionen von Grösse nachgeladener Elemente abhängen z.B. Platzieren von Overlay-Elementen bei grossen Bildern

**Async:** Skripte werden asynchron, sofort geladen und ausgeführt. Reihenfolge wird nicht gewährleistet

**Defer:** Skripte werden nach dem Parsen des Dokuments ausgeführt, Reihenfolge wird gewährleistet

Text

Description automatically generatedA picture containing diagram

Description automatically generated

Chart

Description automatically generated

A picture containing shape

Description automatically generated

## dom-interfaces

**EventTarget:** Objekt kann Events empfangen (addEventListener(), dispatchEvent())

**Node:** Einheitliches Interface für Tree z.B Traversierung von Nodes (childNodes, nextSibling, appendChild(), removeChild())

**Element:** Basis für Elemente im DOM-Tree. Definiert Methoden auf Element-Level z.B .children liefert Elemente, keine Nodes (id, innerHTML, getAttribute())

**HTMLElement:** Repräsentiert HTML-Element (dataset, style, hidden)

**HTMLTableElement:** Methoden um mit Tabelle umzugehen (createCaption(), createTHead())Diagram

Description automatically generated

## dom-manipulation

**Beispiel:**

const newEl = document.createElement('div');  
newEl.appendChild(document.createTextNode('Hello'));  
document.querySelector("#container").appendChild(newEl); Diagram

Description automatically generated

**createDocumentFragment():** Ein temporäres Fragment, welches beim anhängen an Parent entfernt wird

Diagram

Description automatically generated

**createElement vs. innerHTML:** createElement ist Schneller bei kleinen Änderungen. DOM-Referenzen und Event-Handler bleiben erhalten. innerHTML ist wahrscheinlich schneller

**textContent vs. innerText:** textContent gibt alles als Text zurück und berücksichtigt kein CSS und gibt . innerText liefert gerenderten Text und berücksichtigt CSS

## dom-events

Events sind Reaktionen auf Ereignisse (Button wird geklickt, Fenster verkleinert, etc.)

**Beispiel:** document.querySelector("#1").addEventListener("click", () => *alert*('1'));

1. **Capture-Phase:** Event durchläuft DOM-Tree von Wurzel zu Blatt. Jedes Element kann reagieren
2. **Target-Phase:** Event wird auf Target ausgelöst
3. **Bubble-Phase:** Event durchläuft DOM-Tree von Blatt zur Wurzel. Jedes Element kann reagieren

Event-Handler bei Parent Node registrieren, damit nur einer nötig ist, unabhängig von childNodes.

**Eventlistener entfernen:**

document.querySelector('button')  
 .removeEventListener('click', useLightMode);

**Event auslösen:**

document.querySelector('button')  
 .dispatchEvent('click');

Diagram

Description automatically generated

# HttP

**Komponente:** Client und Server tauschen Nachrichten aus, welche aus Header und Body bestehen.

**Request:**

A picture containing text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Get:** Daten abrufen, soll keine Seiteneffekte haben

**Post:** Sendet Daten an den Server. Erzeugt meistens eine Ressource

**Put:** Ersetzt alle Darstellungen der Zielressource durch hochgeladenen Inhalt (idempotent).

**Delete:** Entfernt alle Darstellungen der durch URI angegebenen Zielressource.

**Response:**

**Text

Description automatically generated**

## nodejs

Laufzeitumgebung für server-seitiges JavaScript. JavaScript ausserhalb des Browsers. Verwendet V8-JavaScript Engine aus Chrome. Bringt viele Standardpackages mit.

**Scope:** Node.js kennt file Scope ; window / global heissen unterschiedlich

**NPM:** Packet Manager für JavaScript

**Webserver Beispiel:**

**import** http **from 'http'**;  
**const** host = **'127.0.0.1'  
const** port = 3000  
**const** server = http.createServer((req, res) => {  
 res.**statusCode** = 200  
 res.setHeader(**'Content-Type'**, **'text/plain'**)  
 res.**end**(**'Hello World\n'**)  
})  
server.listen(port, hostname, () => {  
 ***console***.log(**`Server running at http://**${host}**:**${port}**/`**)  
})

# ajax

Mit AJAX (Asynchronous JavsScript and XML) können Web Applikationen Daten asynchron senden und erhalten ohne die bestehende Seite zu beeinträchtigen.

**Vorteil:** Flüssigere Interaktion, Seite ohne Neuladen aktualisieren, Kürzere Zeit bis zum "First Meaningful Paint", da Daten nach Bedarf geladen werden

**Nachteil:** Zurück-Funktion, und URLs müssen separat programmiert werden

**Beispiel fetch Request (then):**

function fetchAndThenLog(fileURL) {  
 *fetch*(fileURL).then(response => {  
 console.log(response)  
 return response.text();  
 }).then(text => console.log(text));  
}  
**Beispiel einfacher Request (await):**  
async function awaitFetchAndLog(fileUrl) {  
 const response = await *fetch*(fileUrl);  
 console.log(response);  
 const responseText = await response.text();  
 console.log(responseText);  
}

Text

Description automatically generated

**Anwendung:** Da JS singlethreaded ist, kann nur etwas auf einmal gemacht werden. Dies macht z.B Animationen problematisch, da diese von Requests blockiert werden können. Um dieses Problem zu lösen, die async Engine hat eine «Queue», welche einen Request nicht sofort ausführt, sondern dann, wenn Zeit dafür besteht.

## promise

Ein Promise ist wie ein Versprechen, sodass man irgendwann eine Antwort erhalten wird, allerdings gibt es keine Garantie für den Wert, er kann alles oder sogar null/undefiniert sein. Dies wird meist verwendet, um einen Wert zu erhalten, bei dem man sich nicht sicher sein kann, dass man ihn erhält. Promise kapselt asynchrone Operationen. Falls Operation unvollständig: pending, falls erfolgreich beendet und Ergebnis verfügbar: fulfilled, ansonsten rejected

async function fetchFromURL() {  
 return new *Promise*((resolve, reject) => {  
 const response = *fetch*(url)  
 .then(res => res.json())  
 .then(json => resolve(json))  
 }).catch(err => reject(err));  
}

## async / await

**Async:** Kennzeichnet eine Funktion, welche await verwendet

**Await:** Pausiert Ausführung bis definierte Funktion resolved ist.

**Beispiel await mit Promises:**

function resolveAfter2Seconds(x) {  
 return new *Promise*(resolve => {  
 *setTimeout*(() => resolve(x), 2000);  
 });  
}  
async function f1() {  
 let x = await resolveAfter2Seconds(10);  
 console.log(x); *// 10*}

**Async Songs mit await:**

async function getAndRenderSongs() {  
 const songs = await dataService.promiseSongs();  
 renderSongs(songs);  
}  
**Async Songs mit then:**  
function getAndRenderSongs() {  
 dataService.promiseSongs()  
 .then(songs => renderSongs(songs));  
}

**Fehlerbehandlung:** Kann mit try-catch gemacht werden

## ajax herausforderungen

**Herausforderungen bei AJAX:** Umgang mit Ladezeiten, unzuverlässiger API-Service, Debouncing, langsamer API-Service bei kontinuierlicher Nutzerinteraktion z.B. Auto-Complete, Server Push

**Suspense State:** UI-Elemente deaktivieren und ein Feedback geben, dass Anfrage läuft

**Fehler oder fehlende Antworten:** Zufällige Meldungen von Fehlern (404, 401, etc.), Nutzer sehen Fehlermeldungen. Abhilfe: Request wiederholen, Fehlermeldung ausgeben etc.

# rest

Ziel ist, den Dienst nach aussen zur Verfügung zu stellen. Andere Geräte können Webservice über Web nutzen.

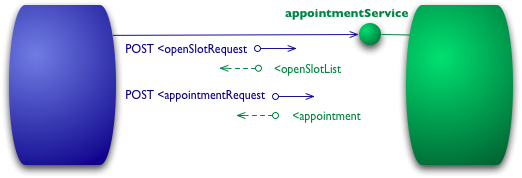
**Eigenschaften:** Trennung von Client und Server. Jeder Request beinhaltet alle benötigten Informationen zur Interpretation. Ergebnisse von Requests können gecacht werden. Erlaubt Einsatz von Schichtenarchitekturen (Load-Balancer, Proxies, Firewall), Selbstbeschreibende Nachrichten, Ressourcen-basierte Schnittstelle.

## richardson’s maturity model

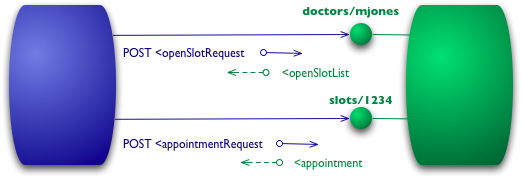
**Graphical user interface, application

Description automatically generated**

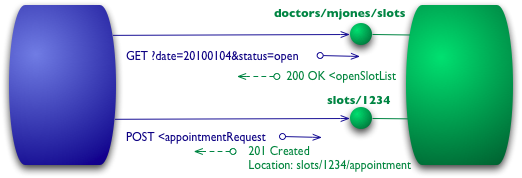
**Level 0:** Maturity Level 0 macht keinen Gebrauch von URI, HTTP-Methoden und HATEOAS-Funktionen. Diese haben einen einzigen URI und verwenden eine einzige HTTP-Methode.



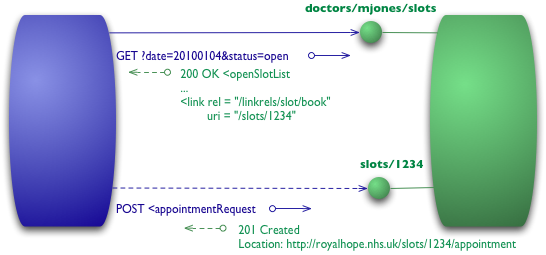
**Level 1:** Maturity Level 1 macht Gebrauch von URI, aber nicht HTTP-Methoden und HATEOAS-Funktionen. Diese Dienste geben jeder im System verfügbaren Ressource einen eindeutigen URI. Ein eindeutiger URI identifiziert eine eindeutige Ressource - und das macht diese Dienste besser als Level 0.



**Level 2:** Maturity Level 1 macht Gebrauch von URI und HTTP-Methoden, aber nicht von HATEOAS-Funktionen. Solche Dienste unterstützen mehrere der HTTP-Verben für jede exponierte Ressource - Dienste zum Erstellen, Lesen, Aktualisieren und Löschen (CRUD). Hier kann der Zustand von Ressourcen, die in der Regel Geschäftseinheiten darstellen, über das Netz manipuliert werden.



**Level 3:** Maturity Level 1 macht Gebrauch von URI, HTTP-Methoden und HATEOAS-Funktionen. Stufe drei ist die ausgereifteste Stufe des Richardson-Modells, die die leichte Auffindbarkeit fördert. Auf dieser Ebene können die Antworten durch die Verwendung von HATEOAS leicht selbstbeschreibend sein.



## resource-oriented architecture

Jede Ressource hat eindeutigen URI (z.B orders/1, books/0-330-25864). Wenn Sub-Ressource ohne Parent nicht existieren kann (z.B customers/1/orders, topics/1/comments).

**Ressourcetyp:** Die gleiche Ressource kann verschiedene Repräsentationen (HTML, JSON, XML, CSV etc.) haben, welche im Accept Request-Header definiert ist.

**Request:**

Graphical user interface, application

Description automatically generated

**Query-Parameter:**

* Filtern (orders?state=active&seller\_id=1234)
* Sortierung (orders?order-by=date&order-direction=asc)
* Pagination (GET /companies?skip=10&take=10)

## http-verben

**Get:** Accept-Header definiert Repräsentation. Query-Parameter für Filter/Selektion. HTTP Status-Codes bei Response beachten.

**Post:** Server generiert URI, Content-Type-Header angeben. Im Response Header Location wird URI der neuen Ressource mitgeteilt.

**Put:** Client definiert URI (im Gegensatz zu POST). Aktualisiert/Erzeugt Ressource. Put-Requests sind idempotent.

**Post vs Put :**

|  |  |
| --- | --- |
| Post | Put |
| POST api/orders/  BODY: {productId: 1234, amount: 2} | PUT api/orders/3  BODY: {productId: 1234, amount: 2} |
| Neue Bestellung erzeugen | Falls nicht vorhanden. wird neue Bestellung erzeugt mit id 3 |
| Server wählt ID und gibt erzeugte URI im Location-Header zurück | Client wählt URL |
| Nicht idempotent | Falls vorhanden und erlaubt wird vorhandene Bestellung überschrieben |
|  | Idempotent |

**Delete:** Ressource löschen

**Patch:** Partielles Update

**Best Practice:** Eine API ist nur so gut wie ihre Dokumentation. Ressource sollten Nomen und keine Verben sein. JSON und kein XML zurückgeben. Statuscodes korrekt einsetzen.

**Response Codes:**

**Table

Description automatically generated**

# usability

**Heuristiken:** Sicherheitsabfragen, Möglichkeit zum Undo bereitstellen, Klare unterscheidbare Beschriftungen, Unterschiedliches sollte unterschiedlich sein

**Wahrnehmung:** Auge hat kleinen „Scharfen Fleck“ (Fovea). Im zentralen Bereich werden Farben und Details wahrgenommen. Im peripheren Bereich werden Bewegungen und Veränderungen wahrgenommen. Statische Informationen im peripheren Bereich

werden leicht übersehen. Aufmerksamkeit kann gelenkt werden (Position, Eye Catcher, Bewegung)

**Übereinstimmung zwischen System und realer Welt:** Sprache der Nutzenden sprechen, Bekannte Begriffe verwenden, Konventionen respektieren, Bezug zur echten Welt herstellen (Affordances, Constraints, Logisches und natürliches Mapping und Metaphern)

Table

Description automatically generated

**Nielsen-Heuristiken:**

1. Sichtbarkeit des Systemstatus
2. Übereinstimmung zwischen System und realer Welt
3. Freiheit und Kontroller der Nutzer/in
4. Konsistenz und Standards
5. Fehlervermeidung
6. Wiedererkennen statt erinnern
7. Flexibilität und effiziente Nutzung
8. Ästhetik und minimalistische Gestaltung
9. Hilfe beim Erkennen und Beheben von Fehlern
10. Hilfe und Dokumentation

**Sichtbarkeit des Systemstatus:** Zustand des Systems anzeigen (Was tut es gerade? Wie lange dauert es noch?)

**Übereinstimmung zwischen System und realer Welt:** Sprache des Nutzer verwenden. Bekannte Begriffe verwenden und Konventionen respektieren. Bezug zur echten Welt herstellen mittels Affordances, Contraints, logisches und natürliches Mapping und Metaphern.

**Konsistenz und Standards:** Begriffe konsistent verwenden (Sichern = Speichern?, Schliessen = Weiter?) und Konventionen beachten (Links unterstreichen, Buttons sind klickbar, etc.)

**Fehler vermeiden:** Undo behebt Fehler beim Experimentieren, hilft Sicherheitsabfragen zu meiden

**Zentrales/Peripheres Sehen:**

**A picture containing graphical user interface

Description automatically generated**

## affordances

**Gibson:** Natürlich wahrnehmbare Nutzungsmöglichkeit eines Geräts

**Norman:** Wahrgenommene affordance: Touchscreen = Tap, Swipe. Maus = Clicken, Draggen

**Formulare:** Ein Textfeld sollte wie ein Textfeld aussehen. Leere Felder signalisieren "Füll mich aus", weil sie leer sind. Checkboxes, Radiobuttons, Links korrekt verwenden. Gute Beschriftungen mit entsprechenden Hinweisen.

**Selection Controls:**

**Diagram

Description automatically generated**

**Affordance:** Natürlich wahrnehmbare Nutzungsmöglichkeit eines Geräts Affordances

ermöglichen: Konventionen befolgen, Konsistente Verwendung, Beschriftung beschreibt Aktion, Metaphern mit Vorsicht verwenden

## visuelles design

**Allgemein:** Beeinflusst Lesbarkeit, Lenkt Aufmerksamkeit, Schafft Vertrauen und Glaubwürdigkeit, Stärkt die Marke

**Gestaltgesetze:**

Diagram

Description automatically generated with medium confidence

**Gestalt der Ähnlichkeit:** Elemente die ähnlich aussehen werden als Gruppe interpretiert. Farben, Grössen, Bewegungen, Formen. Strukturierung stärker: Grad der Ähnlichkeit der Elemente. Kontrast zu benachbarten Elementen. Anzahl der ähnlichen Eigenschaften

**Gesetz der Nähe:** Elemente die räumlich nah beieinander liegen, werden durch unsere Wahrnehmung als eine Gruppe interpretieren. Benachbarte Elemente werden gruppiert (z.B. Spalten/Zeilen). Benachbarte Elemente sollten auch zusammengehören. Abständen bewusst verwenden

**Zusammenfassung:** Idee: Informationsverarbeitung vereinfacht wahrgenommene Gegenstände nach bestimmten Regeln. Ziel: Effizienz und Reduktion der Komplexität. Erklären menschliche Wahrnehmung. Heuristiken die auch widersprüchliche Interpretationen zulassen.

## farb-design

**Grundlagen:** Wenige Farben (< 6, besser: 2 + 1). Standards/Normen (z.B. Styleguides), Kulturelle Aspekte, Ausreichender Kontrast

**Physiologische Aspekte:** Farbfehlsichtigkeit, Schlechte Farbkombinationen, Seheinschränkungen beachten

## usability tests

**Wieso:** Probleme erkennen, Verbesserungsmöglichkeiten identifizieren, User kennenlernen

Diagram

Description automatically generated

**Elemente eines Usability-Tests:**

A picture containing diagram

Description automatically generated

# Beispiele

## js math

"4" / "2" *//=> 2: Number*  
"4" – "2" *//=> 2: Number*  
"4" \* "2" *//=> 8: Number*  
"4" + "2" *//=> '42': String*  
10 \* 3 + "px" *//=> '30px': String*  
8 \* "1px" *//=> NaN: Number*  
"px" + 1 -2 *//=> NaN: Number*  
"3px" + 3 \* 2 + "3px" *//=> '3px63px': String*  
"foo" ++ "abc" *//=> 'fooNaN': String*  
"2" - -1 *//=> 3 :Number*  
[] + [] *//=> '': String*  
[] + {} *//=> '[object Object]': String*

{} + [] *//=> 0: Number*

{} + {} *//=> '[object Object][object Object]': String*  
[] == false *//=> true*  
[] === false *//=> false*

[] == 0 *//=> true*

[] == "0" *//=> false*  
"" == false *//=> true*  
"" === false *//=> false*  
0 == "0" *//=> true*  
0 === "0" *//=> false*  
null == undefined *//=> true*  
null === undefined *//=> false*  
[1,2] == "1,2" *//=> true*  
[1,2] === "1,2" *//=> false*  
NaN == NaN *//=> false*  
NaN === NaN *//=> false*  
(1-"i") == (1-"i") *//=> false: (NaN == NaN)*  
[] == ] *//=> true*  
[] === ![] *//=> false*  
false === false *//=> true*  
4 === 4 *//=> true*  
false === true *//=> false*  
true + true + true == 3 *//=>true*  
true – true *//=> 0*  
true == 1 *//=> true*  
true === 1 *//=> false*  
a={0,1,2}  
a[3] === undefined *//=> true*  
a[3] == null *//=> true*  
a[3] === null *//=> false*  
*parseInt*("4k", 10) *//=> 4*  
*parseInt*("4k", 2) *//=> NaN*  
*parseInt*("4k", 30) *//=> 140: = 4\*30^1 + k\*30^0*  
*Number*("4k") *//=> NaN*  
*Number*("4") *//=> 4*

## single page application

const postsRESTServerURL = 'http://localhost:3000/';  
const postsRoute = 'posts';  
const commentsRoute = 'comments';  
  
async function getJson(url) {  
 const response = await *fetch*(url);  
 return response.json();  
}  
  
async function postJson(url, json) {  
 const response = await *fetch*(url, {  
 method: 'post',   
 headers: {  
 'Accept': 'application/json',   
 'Content-Type': 'application/json',  
 },   
 body: JSON.stringify(json),  
 });  
 return response.json();  
}

async function getPosts() {  
 return await getJson(postsRESTServerURL + postsRoute);  
}  
  
async function getComments() {  
 return await getJson(postsRESTServerURL + commentsRoute);  
}  
  
async function submitPost(post) {  
 return await postJson(postsRESTServerURL + postsRoute, post);  
}  
  
function createPostCommentsHtml(comments, postId) {  
 const postComments = comments  
 .filter(comment => comment.postId === postId)  
 .map((comment) => `<li>${comment.body}</li>`)  
 .join('');  
 return (postComments) ? `<ul>${postComments}</ul>` : '';  
}  
async function renderPosts() {  
 const posts = await dataService.getPosts();  
 const comments = await dataService.getComments();  
 const postsHtml = posts  
 .filter(post => post.title)  
 .map((post) =>  
 `<li data-post-id="${post.id}">(${post.id})   
 <em>${post.title}</em>   
 (${post.author})  
 ${createPostCommentsHtml(comments, post.id)}  
 </li>`)  
 .join('');  
 postsList.innerHTML = postsHtml;  
}

## css stuff

**Background:**

background-color: red;  
background-image: "/path";   
*/\* can also be an url -> url(https: //...) \*/*background-size: cover|auto|length|percentage|contain;   
*/\* cover: resizes to parent , auto: size of image , contain: resize image to fit inside parent , initial: fixed x,y value \*/*background-position: x-value y-value;   
*/\* top right , top center , center center , ... Can also be x% y% default 0% 0%. \*/*background-attachement: scroll|fixed|local;   
*/\* local: scrolls with element , scroll is default. this decides if the background scrolls with the page or not \*/*

background-origin: padding-box|content-box|border-box;   
*/\* Used to define where the background image starts inside a box , only works when background-attachement is not fixed .\*/*background-repeat: repeat|repeat-x|repeat-y|no-repeat|space|round;   
*/\* useless shit \*/*

**Borders:**

border: 4px solid red;  
border: border-width border-style border-color;  
border-style: none|hidden|dotted|dashed|solid|double|groove|ridge|inset|outset|initial|inherit;  
*/\* styles like ridge , groove , inset and outset use 3D effects , hard to describe. \*/*border-radius: 4px 4px 4px 4px;  
*/\* border-radius: 5px; also possible. borders can be addressed individually by specifying a direction: \*/*border-bottom: 4px solid red;  
border-collapse: separate|collapse|initial|inherit;  
*/\* collapse means borders will merge in boxes. seperate will draw borders for both elements. \*/*border-spacing: 4px;  
*/\* Spacing between borders if the border-collapse is enabled!\*/*

**Margin/Padding:**

margin: top right bottom left;   
margin: 10px 10px 10px 10px;   
*/\* == margin: 10px; \*/*padding: top right bottom left;   
padding: 10px 10px 10px 10px;  
*/\* margin-right padding-left etc work as well. \*/*

**Text:**

text-align: left|right|center|justify;  
*/\* justify is text as a block. \*/*text-align-last: left|right|center|justify;  
*/\* The last line can be configured indiidually. \*/*text-justify: auto|inter-word|inter-character|none;  
*/\* only used when text-align is set to justify. inter-word increases spacing between words, inter-character for characters.\*/*text-decoration: text-decoration-line text-decoration-color text-decoration-style text-decoration-thickness;  
text-decoration-line: none|underline|overline|line-through;  
text-decoration-style: solid|double|dotted|dashed|wavy;  
text-decoration-thickness: auto|from-font|value;  
text-indent: value;  
text-overflow: clip|ellipsis|string;  
*/\* clip will just cut the string off , ellipsis ends with ..., a string will display itself. \*/*text-shadow: h-shadow v-shadow blur-radius color|none;  
text-transform: none|capitalize|uppercase|lowercase;

**Font:**

font: font-style font-variant font-weight font-size/ line-height font-family |caption|icon|menu|message-box|small-caption|status-bar;  
*/\* use the font used by this element \*/*font-style: normal|italic|oblique;  
*/\* oblique seems to be the same as italic. \*/*font-variant: normal|small-caps;  
font-weight: normal|bold|bolder|lighter|number;  
*/\* number in 100 steps -> 100 ,200 ,300 ,... ,900 \*/*font-size: medium|xx-small|x-small|small|large|x-large| xx-large|smaller|larger|length;  
line-height: normal|number|length;   
font-family: family-name|generic-family;   
*/\* generic-family, example: Isoveka \*/*

## html tags

**Image:** empty tag, embedded/interactive

<img src="path to source" alt="alternative text">

**Input:** empty tag, interactive

<input type="text" placeholder="something">

**Form with js function:** interactive

<form onsubmit ={() => handleSubmit () }>*<!--Inputs and buttons. -->* </form>

**Headertext:** heading

<h1>Some text to use as header </h1> *<!--h1 ,h2,h3,h4 ,h5,h6-->*

**Paragraph:** phrasing

<p>Some text to display </p>

**Ordered List:** flow

<ol>  
 <li>element1 </li>  
 <li>element2 </li>  
 <li>element3 </li>  
</ol>

**Hyperlink:** interactive

<a href="htts: // dashie.org">Shitgaem </a>

**Bold text:** phrasing

<b>this text is bold , or maybe not.</b>

**Italic text:** phrasing

<em>This text is italic , or maybe not</em>

**Div:** flow

<div>try centering it!</div>

**Navigation Bar:** sectioning

<nav>  
 <a href="/html/">HTML</a>  
 <a href="/css/">CSS</a>  
 <a href="/js/">JavaScript </a>  
 <a href="/cpp/">C++</a>  
</nav>

**Bar on the side:** sectioning

<aside >Something to display on the side</aside>

**Article:** sectioning

<article >*<!-- other elements -->*</article >

**Section:** sectioning

<section >*<!-- should always have a heading! -->*</section>  
*<!-- The difference to article is that section is not independent!  
it always has a heading. -->*

**ol vs ul vs dl:**

![A screenshot of a phone

Description automatically generated with low confidence<ol>  
 <li>a</li>  
 <li>b</li>  
 <li>c</li>  
</ol>  
<ul>  
 <li>a</li>  
 <li>b</li>  
 <li>c</li>  
</ul>  
<dl>  
 <dt>a</dt>  
 <dd>b</dd>  
 <dt>a</dt>  
 <dd>b</dd>  
</dl>