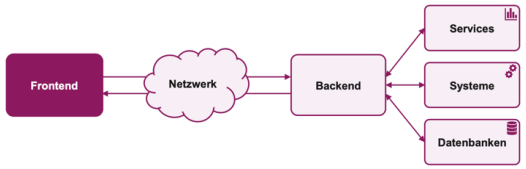
Architektur Webanwendung



HTML

- doctype
- **html** Root-Element
  z.B. Sprache (für Übersetzer / Screen-Reader)
- **head** Metadaten
  Nur 1 head-Element
  Daten über das Dokument
  werden nicht dargestellt
- **body** Sichtbarer Inhalt der Webseite
  nur 1 body-Element

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <link rel="icon"href="ost-favicon-32x32.png">
        <title>hello</title>
    </head>
    <body>
        <!--this is a comment-->
        <h1>Hello World</h1>
        <input type="password" value="1234">
    </body>
</html>
```

- Auszeichnungssprache / Markup Language
- beschreibt **Struktur** eines Dokuments
- Trennung **Inhalt** und **Darstellung**

HyperText Markup Language

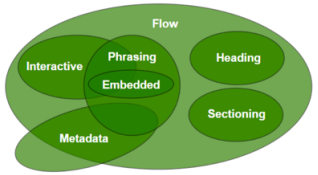
Hypertext	Verlinkung von Dokumenten möglich
Markup	Semantische Auszeichnung von Inhalten (`h1, p, footer, ...`)
Language	Definierte Syntax und Bedeutung

- HTML-Dateien werden decodiert, in DOM übersetzt und vom Browser gerendert
- HTML-Code prüfen → W3C-Validator

Metadaten

```
<head>
    <meta charset="UTF-8"> <title>Title</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="icon" href="ost-favicon32x32.png">
    <meta name="author" content="name, frieder.loch@ost.ch">
    <meta name="viewport" content="width=device-width,
          initial-scale=1">
    <meta name="description" content="A description of the page">
    <meta name="robots" content="index" />
</head>
```

Kategorien

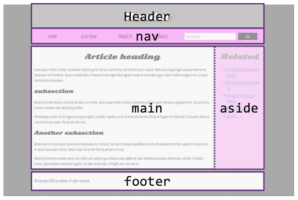


Elemente mit Kategorie und Semantik

Element	Semantik
`<h1> - <h6>`	Überschriften; h1 höchste Stufe
`<p>`	Absatz
`<blockquote>`	Zitat
` `	Semantischer Umbruch
`<b>`	„Bring Attention To“: Wichtig für User
`<code>`	Programm-Code
`<img>`	Bild
`<em>`	Hervorhebung anzeigen
`<strong>`	Starke hervorhebung anzeigen
`<mark>`	Relevanz indizieren
`<cite>`	Name eines Werks, Buch markieren
`. <dfn>`	Definition eines Begriffs markieren

Seitenstruktur mit semantischen Elementen

```
<body>
  <header>
    <nav>
  <main>
    <header>
    <nav>
    <article>
    <aside>
  <footer>
```



Semantisch / Nicht-Semantisch

Semantik: Zuordnung von Bedeutung zu Texten

Schlecht: "div-Soup"

```
<!-- -->
<div class="h1">The
 <div class="em">super duper</div>
   shopping list</div>
<div class="ul">
 <div class="li">Milk</div>
 <div class="li">Cheese
  <div class="ul">
   <div class="li">Blue cheese</div>
   <div class="li">Feta</div>
  </div>
 </div>
</div>
<!-- -->
```

Besser: semantisches HTML

```
<!-- -->
<h1>The <em>Super Duper</em> Shopping List</h1>
<ul>
 <li>Milk</li>
 <li>Cheese
  <ul>
   <li>Blue cheese</li>
   <li>Feta</li>
  </ul>
 </li>
</ul>
<!-- -->
```

`article`-Element

In sich geschlossener Abschnitt eines Dokuments, Seite oder Anwendung die unabhängig verteilt oder wiederverwendbar sein soll.

```
<article>
    <header>
        <h2>Jurassic Park</h2>
    </header>
    <section>
        <p>Way too scary for me.</p>
        <footer>
            <p>Posted on <time datetime="2020-01-01
                    00:01">January 1</time> by Lisa.</p>
        </footer>
    </section>
    <!-- -->
    <footer>
        <p>Posted on <time datetime="2021-01-16 19:00">January
                16</time> by Staff.</p>
    </footer>
</article>
```

`section`-Eelement

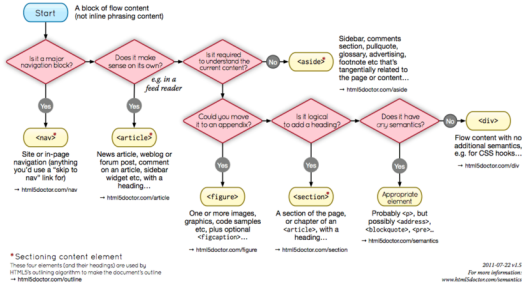
Allgemeine Unterteilung des Dokuments, z.B. thematische Gruppierung des Inhalts mit Überschrift.
→ weniger unabhängig als `article`

```
<main>
    <h1>Star Wars</h1>
    <section>
        <!-- Inhalte -->
    </section>
    <section id="user-reviews">
        <article>
            <!-- Inhalt des Artikels -->
        </article>
    </section>
</main>
```

`figure`-Element

```
<figure>
    <img src="/media/cc0-images/elephant.jpg" alt="Elephant at
         sunset">
    <figcaption>An elephant at sunset</figcaption>
</figure>
```

Auswahl des richtigen Elements



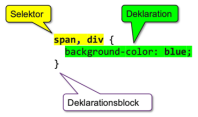
Mögliche Fragen

- Wieso sollte ich semantisches HTML verwenden?
  Leichtere Entwicklung: erhalten einige Funktionen kostenlos, verständlicheres Markup
  Mobile-Freundlich: Geringere Dateigrösse, leichter responsive zu machen
  SEO: Suchmaschinen gewichten Schlüsselwörter in Überschriften, Links, . . . höher
  Zugänglich: Semantisches Markup ist mit Screenreader deutlich einfacher zu verwenden
- Wie kann ich eine gegebene Webseite mit semantischen HTML auszeichnen?
  mit Tags: nav, article, figure, aside, section
- Wo ist Tag Omission möglich?
  img (must not have end tag)
  p (start tag required)
  html
  input (must not have end tag)

CSS

- Inhalt von Darstellung trennen
- Unterschiedliche Styles für unterschiedliche Ausgabemedien
- Styles auf Elementgruppen anwenden
- Regel besteht aus Selektoren und Deklarationsblock

Syntax



CSS-Datei in HTML laden

```
<link rel="stylesheet" href="file.css" />
```

Selektoren

Selektoren sind Case-Sensitiv! (id und class Attribute klein schreiben, zusammengesetzte Worte: kebab-case)

Typ	Selektiert **alle** Elemente vom angegeben Typ
ID	Selektiert **alle** Elemente mit angegebener ID
Klassen	Styling ähnlicher Elemente • `.recent`: Elemente mit dieser Klasse • `.recent.explicit`: Elemente mit beiden Klassen • `article.recent`: article-Elemente mit Klasse recent **Tipp:** Semantische Klassennamen wählen

Pseudo-Elemente

Styling von **bestimmten Teilen** von selektierten Elementen
Element
`::first-line`
`::after / ::before`
`::selection`

Pseudo-Klassen

Selektor-Typ	Beispiel
`:nth-child()`	**`span:nth-child(1)`** **`span:nth-child(odd)`** **`span:nth-child(event)`**
`:first-child`	**`span:first-child`**
`:last-child`	**`span:last-child`**
`:hover`	
`:visited`	
`:active / :focus`	

```
.recent
    span
    span <--
          nth-child
    span
```

```
.recent
    span <--
          first-child
    span
    span
```

```
.recent
    span
    span
    span <--
          last-child
```

Kombinatoren

Selektor-Typ	Beispiel
Nachfahren (decendant)	`.recent div`
Direkte Nachfahren (child)	`.recent > div`
Angrenzende Geschwister (**next** siblings)	`.recent > p:first-of-type + p`
Alle Geschwister (**subsequent** siblings)	`.recent > p:first-of-type p`

```
.recent
    div          <-- decendant
        div      <-- decendant
```

```
.recent
    div          <-- child
        div
```

```
.recent
    p
    p   <-- +-Selector
    p
    p
```

```
.recent
    p
    p   <-- ~-Selector
    p   <--
    p   <--
```

Kaskade

- Mehrere Deklarationen betreffen 1 Element
- Kaskade löst Konflikte auf
- Kaskade betrachtet folgende Komponenten:
  Woher kommt Style?
  Spezifität des Selektors
  Reihenfolge der Styles
  `!important`

Priorität:

1. Autor:in (Externe / Interne / Inline Styles)
2. Benutzer:in (Browser-Einstellungen)
3. Browser (Default-Werte zur Darstellung)

- Innerhalb der Punkte: Sortieren nach Spezifität
- Bei gleicher Spezifität: **Später** deklarierte Eigenschaft gewinnt
- `!important`-Deklarationen der Benutzer:in gewinnen immer

Spezifität

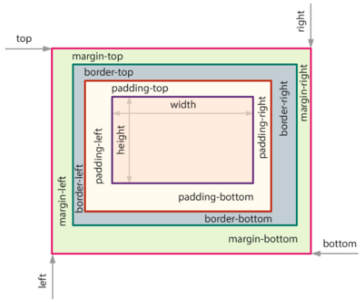
- Regeln werden nach Spezifität gewichtet
- Vier Zähler (A, B, C, D) mit Startwert 0
  a++ Inline-Styles
  b++ ID-Selektoren
  c++ Klassen-Selektoren, Pseudoklassen (: . . .) und Attribute ([. . .])
  d++ Typ-Selektoren und Pseudoelemente (:: . . .)
- Bei gleicher Spezifität: Reihenfolge

Nico Fehr | Ostschweizer Fachhochschule

Selektor	a	b	c	d	=
h1	0	0	0	1	0001
ul li	0	0	0	2	0002
#identifier	0	1	0	0	0100
h1#identifier	0	1	0	1	0101
style=„“	1	0	0	0	1000
p:first-child	0	0	1	1	0011
#editor p	0	1	0	1	0101

- Universalselektor "*"wird ignoriert
- Pseudoklasse `:not()` wird ignoriert (Selektoren innerhalb Klammern nicht)

Box Model



Element besteht aus:

- Content: Inhalt
- Padding: Innenabstand
- Border: Rahmen
- Margin: Aussenabstand

Usability

Heuristiken

Heuristik: Sammlung von erprobten „Daumenregeln“

- Sicherheitsabfragen
- Möglichkeit zum Undo bereitstellen
- Klare, unterscheidbare Beschriftungen
- Unterschiedliches sollte unterschiedlich sein

Wahrnehmung

Aufmerksamkeit kann **gelenkt** werden

Change Blindness

- Veränderungen, die in einem Bereich auftreten, der vom Aufmerksamkeitsfokus entfernt ist, werden eher übersehen
- Passiert wenn visuelles Signal für Änderung fehlt
- Aufmerksamkeit auf Änderungen lenken

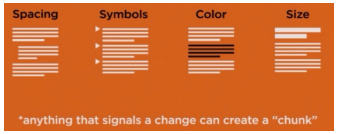
Lesen

„Web Users don’t read, they scan“

Augenführung durch:

- Grafiken
- Textgrössen
- Farben

→ Informationen **hierarchisch** darstellen
Methoden zur Strukturierung



Unterstützen verschiedener Rezeptionsformen

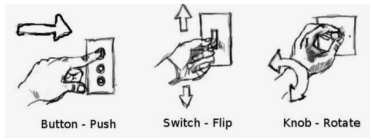
Rezeption	Gestaltung
Scannen	Akzente • Überschriften • Schlüsselwörter, -bilder • Links / Suchhilfen
Skimmen	Zusammenfassen • kurze Absätze • Listen / Tabellen • Info-Grafik
Lesen	Details • Längere Seiten mit Fliesstext • Format zum Ausdrucken

Sehfähigkeit

- Auge hat kleinen „Scharfen Fleck“ (*Fovea*)
- Statische Informationen im peripheren Bereich werden leicht übersehen

Bereich	Was wird wahrgenommen?
zentraler Bereich	Farben und Details
peripherer Bereich	Bewegungen und Veränderungen

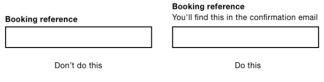
Affordance von UI-Elementen



Affordance: Natürlich wahrnehmbare Nutzungsmöglichkeit eines Geräts

Affordances ermöglichen:

- Konventionen befolgen
- Konsistente Verwendung
- Beschriftung beschreibt Aktion
- Metaphern mit Vorsicht verwenden



The booking reference field without hint (left) and with hint (right).

Gute Beschriftungen mit entsprechenden Hinweisen

Gibson	Natürlich wahrnehmbare Nutzungsmöglichkeit eines Geräts
Norman	*Wahrgenommene* affordance Touchscreen → Tap, Swipe, ... Maus → Clicken, Draggen, ...

10 Nielsen Usability Heuristiken

1 Sichtbarkeit des Systemstatus
2 Übereinstimmung zwischen System und realer Welt
  Bekannte Begriffe verwenden (Sprache der User)
  Konventionen respektieren
3 Freiheit und Kontrolle der User
4 Konsistenz und Standards
  Begriffe konsistent verwenden
  Links unterstreichen, Buttons klickbar
5 Fehlervermeidung
  Undo → behebt Fehler beim Experimentieren
  Genügend grosse „Touch-Targets“
  Vorzeitige Evaluation von Datumsfeldern
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 zeigen
Zeigt Möglichkeiten der Interaktion

- Welche *Objekte* können selektiert werden
- Welche *Aktionen* können ausgeführt werden
- Welche *Navigationsmöglichkeiten* existieren

Visuelle Gestaltung Gestaltungsgesetze

- Beeinflusst lesbarkeit
- Lenkt Aufmerksamkeit
- Schafft Vertrauen und Glaubwürdigkeit
- Stärkt die Marke

Gesetze der Zusammengehörigkeit

Welche Elemente werden als zusammengehörig wahrgenommen?

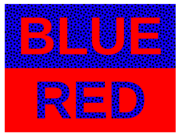
Gesetz der Ähnlichkeit

- Elemente die ähnlich aussehen werden als Gruppe interpretiert
- Ähnliches Zusammenfassen
  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 als Gruppe interpretiert
- Benachbarte Elemente gruppieren (Spalten / Zeilen)
- Benachbarte Elemente sollten zusammengehören
- Abständen bewusst werden

Farben



Chromostereopsie

- Wenige Farben (< 6, besser 2 + 1)
- Standards / Normen (z.B. Styleguides)
- Kulturelle Aspekte !
- Ausreichender Kontrast
- **Pyhisologische Aspekte**
  Farbfehlsichtigkeit
  Schlechte Farbkombis
  Seheinschränkungen beachten

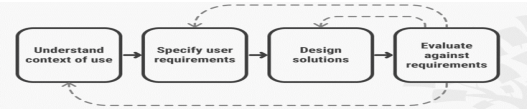
Kulturelle Aspekte

Rot nicht immer „Stop / Achtung“
Grün nicht immer „Gut / Weiter“

Farbfehlsichtigkeit

9% aller Männer, 0.8% aller Frauen haben Rot-Grün Sehschwäche

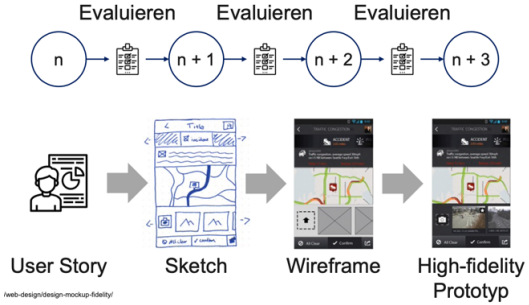
Usability Testing



- Probleme erkennen
- Verbesserungsmöglichkeiten identifizieren
- User kennenlernen

Flow of Information

Iterative Entwicklung



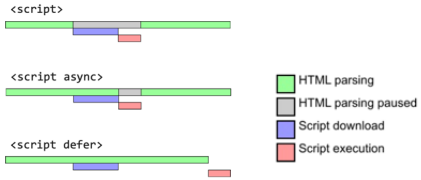
DOM

DOM-Lifecycle

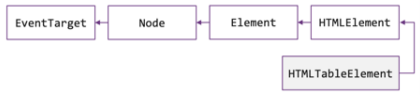
Ladezustand des Dokuments

1. `Loading`
   Dokument lädt noch
2. `Interactive`
   Dokument geladen/geparst
   Ressourcen (Scripts, Bilder / Video, Stylesheets) werden noch geladen
   Event: DOMContentLoaded
3. `Complete`
   Dokument und Ressourcen geladen
   Event: load

async / defer



DOM-Interfaces



Knoten im DOM-Tree implementieren Interfaces

EventTarget	Wichtige Methoden • addEventListener • removeEventListener • dispatchEvent Objekt kann Events empfangen
Node	• Einheitliches Interface für Tree • z.B. finden und Traversieren der Nodes Wichtige Properties • childNodes • firstChild • nextSibling Wichtige Methoden • appendChild() • removeChild()
Element	• Basis für Elemente im DOM-Tree • Definiert Methoden auf Element-Level Wichtige Properties • id • className / classList • innerHTML Wichtige Methoden • getAttribute()/setAttribute()/toggleAttribute() • closest()
HTMLElement	Repräsentiert HTML-Element Wichtige Properties • dataset • style • hidden
HTMLTableElement	Methoden um mit Tabelle umzugehen Wichtige Methoden • createCaption() • createTFoot() • createTHead()

DOM-Navigation Element Node

Node besitzt Eigenschaften zur Navigation im DOM

- .parentElement
- .childNodes
- .children
- .firstChild
- .firstElementChild
- .nextSibling
- .nextElementSibling

HTML-Elemente sind auch Knoten

```
document.querySelector("#progress-div").parentElement;
```

window

- Fenster, in dem das Dokument dargestellt wird
- Stellt globale Objekte zur Verfügung
  - console
  - document
  - HTMLDocument
- Globale Variablen liegen auf window

DOM-Selektion

getElementBy

- document.getElementById (<idString>)
- document.getElementByName (<nameString>) // Form-Element
- <searchRootElement>.getElementsByClassName (<singleClassString>) // HTMLCollectionOf<Element>
- <searchRootElement>.getElementsByTagName (<tagString>)

querySelector

- <searchRootElement>.querySelector (<selectorString>) // first match
- <searchRootElement>.querySelectorAll (<selectorString>) // NodeListOf<HTMLElement>
- <searchRootElement>.closest (<selectorString>) // first matching ancestor
- <searchRootElement>.matches (<selectorString>) // boolean

DOM-Manipulation

createElement & appendChild

- Schneller bei kleinen Änderungen
- DOM-Referenzen bleiben erhalten
- Event-Handler bleiben erhalten

```
// 1
const newEl = document.createElement('div');
newEl.appendChild(document.createTextNode('Hello'));
document.querySelector("#container").appendChild(newEl);

// 2
const newEl2 = document.createElement('div');
newEl2.innerText = 'World';
document.querySelector("#container").appendChild(newEl2)
```

createDocumentFragment

```
const fragment = document.createDocumentFragment();
parent.appendChild(fragment);
```

innerHTML

- Wahrscheinlich schneller als createElement
- Code ist lesbarer

```
// Inhalt schreiben
const c = document.querySelector("#container");
c.innerHTML = '<div>Changed</div>';
c.innerHTML = `<div>${content}</div>`;

// Inhalt loeschen
c.innerHTML = '';

// An bestimmter Position einfuegen
insertAdajcentHTML(position, text)
```

Umgang mit CSS-Klassen

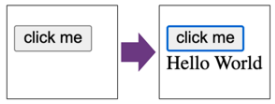
- <element>.className
- Klassen als String, getrennt mit Space
- <element>.classList
- Klassen als DOMTokenList
- DOMTokenList bietet Hilfsfunktionen
- add / remove / toggle / contains / replace

DOM Attribute

- setAttribute('attributeName', 'newAttributeValue')
- setzt beliebige Werte beliebiger Attribute
- setzt auch entsprechende Property
- removeAttribute('attributeName')
- Löscht Attribut

DOM-Events


```
<button id="btnClickMe">click me</button>
<div id="container"></div>
<script>
  const btn =
    document.querySelector("#btnClickMe");
  const container =
    document.querySelector("#container");
  const eventListener = () => {
    container.appendChild(
      document.createTextNode("Hello World"));
  };
  btn.addEventListener("click", eventListener);
</script>
```



- HTMLElement bietet Events, auf die EventListener registriert werden können
- Mit addEventListener werden EventListener registriert
- EventListener wird beim Eintreten des Events aufgerufen

addEventListener

```
target.addEventListener(type, listener[, options])
```

Type
- Name vom Event

Listener
- EventListener Funktion
- Inline oder separat definiert

Options-Objekt
- capture: Reagiert auf Events in der Capture-Phase (boolean)
- once: Listener wird automatisch nach erster Aktivierung entfernt (boolean)
- passive: Für Performance-Optimierungen (boolean)

EventListener registrieren

```
document.querySelector("#1").addEventListener("click", () =>
      alert('1')); // 1

document.querySelector("#2").onclick = () => alert('2'); // 2

<button onclick="alert('3')">3</button> // 3
```

Event-Objekt

Folgende Properties & Methoden

- target
  Element von dem Event ausging
- currentTarget
  Zeigt auf Element, das Event-Listener registriert hat
- preventDefault()
  Verhindert Default-Aktionen
- stopPropagation()
  Event von Bubbling/Capturing abhalten

Spezifische Typen
- MouseEvent
- WheelEvent
- InputEvent

KeyboardEvent-Objekt

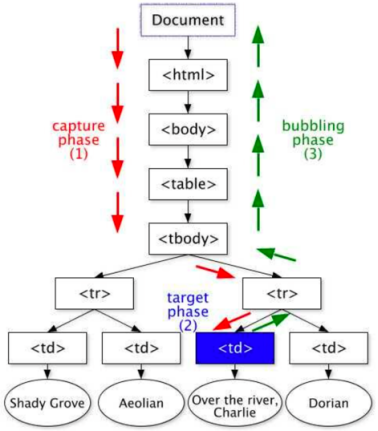
EventListener erhält Event-Objekt mit weitern Infos

- change: Was wurde geändert?
- keydown: Welche Taste gedrückt?
- ctrlKey: War Control Taste gedrückt?

```
document.querySelector("input").addEventListener("keydown",
      (event) => {
    console.log(event.key);
})
```

Event-Bubbling

- Event-Handler in Parent Element definieren
- click-Events aller Children in bubble-Phase bearbeiten
- Nur ein Handler nötig, unabhängig von Anzahl der Bilder



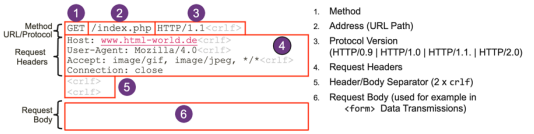
1. Capture-Phase
   Event durchläuft DOM-Tree von Wurzel zum Blatt
   Jedes Element kann reagieren (muss aber nicht)
2. Target-Phase
   Event wird auf Target ausgelöst
3. Bubble-Phase
   Event durchläuft DOM-Tree vom Blatt zur Wurzel
   Jedes Element kann reagieren (muss aber nicht)
   Nicht jedes Event durchläuft Bubble-Phase

HTTP, REST, AJAX

HTTP

- Anfordern von Ressourcen über Netzwerk
- **Client** und **Server** tauschen **Nachrichten** (Request / Response) aus
- Nachrichten bestehen aus **Header** und **Body**
- **Request:** Client → Server
  Typ: GET, POST, PUT, DELETE, PATCH
  Headers
- **Response:** Server → Client
  Status Code: 404, 200, ...
  Body

Request

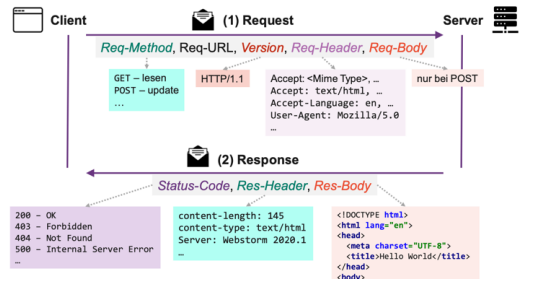


Response

Response Status Codes

Code	
1xx	Informational
2xx	**Successful** 200 OK 201 Created 204 No Content
3xx	**Redirection** 301 Moved Permanently
4xx	**Client Error** 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
5xx	**Server Error** 500 Internal Server Error 505 HTTP Version Not Supported
9xx	Non-Standard Codes

Request → Response



Request-Methoden

GET	Daten abrufen, soll keine Seiteneffekte haben
POST	Sendet Daten an Server, erzeugt meist neue Ressource (nicht-idempotent)
PUT	Ersetzt alle Darstellungen der Zielressource durch hochgeladenen Inhalt (idempotent)
DELETE	Entfernt alle Darstellungen der durch URI angegebenen Zielressource

Request absetzten JS

```
async function fetchFromURL() {
  try {
    const response = await fetch('result.json', { method: 'GET'});
    if(response.ok) {
      const content = await response.text();
      console.log(content);
    }
  } catch(e) {
    // Fehlerbehandlung
  }
}
```

Node.js

HTML-Server

```
function requestHandler(req, res) {
    if (req.url === '/favicon.ico') {...}
    const parsedURL = new URL(req.url,
          `http://${req.headers.host}`);
    const responseHtml = createResponseHtml(parsedURL);

    res.setHeader('Content-Type', 'text/html');
    res.statusCode = (responseHtml) ? 200 : 404;
        res.end(responseHtml);
}
function createResponseHtml(parsedURL) {
    if (parsedURL.pathname === '/hello') {
        return `<h1>Hello</h1> <p>${parsedURL.href}</p>`;
    }
    return '';
}
```

HTML File Server mit node-static

```
import http from 'http';
import nstatic from 'node-static';

const PORT = 8080;
const fileServer = new nstatic.Server('./pub');

function requestHandler(req, res) {
    fileServer.serve(req, res);
}

const server = http.createServer(requestHandler);
server.listen(PORT, () => console.log('Node listening on Port ',
      PORT));
```

Promises

„A sign, or a reason for hope that something may happen, especially something good.“

Promise-Objekt stellt möglichen Abschluss (oder Misserfolg) einer asynchronen Operation und den daraus resultierenden Wert dar.

Die **asynchrone Methode** gibt ein Versprechen (Promise) zurück, den Wert **in der Zukunft** zu liefern

Daten von URL laden

```
async function fetchFromURL() {
    return new Promise ((resolve, reject) => {
        const response =
              fetch('https://stone.sifs0005.infs.ch/ranking')
            .then(res => res.json())
            .then(json => resolve(json))
    }).catch(
        err => reject(err)
    );
}
```

Anwendung in JavaScript

Promise kapselt `async`-Operation

- **pending:** Operation aktiv
- **fulfilled:** Operation erfolgreich beendet und Ergebnis verfügbar (Settled)
- **rejected:** Operation gescheitert (Settled)

Möglichkeit auf Zustandsänderungen zu reagieren

- `.then(..)`
- `.catch(..)`
- `.finally(..)`

Konstruktor

Argument: Executor-Funktion: `Fn(resolveFn, rejectFn) => void`

```
new Promise((resolve, reject) =>
    // Synchronous setup possible here
    >>>async fn<<<(..., (...callbackArgs) => {   } Zu "wrappende" async-Funktion aufrufen
    if (error) {
        reject(error);          } Aufruf von reject (wo sinnvoll)
    } else {
        resolve(value);         } Aufruf von resolve mit Ergebnis
    }
})
```

Promises - Lifecycle

```
function resolveAfter2Seconds(x) {
  return new Promise(
    resolve => {
      setTimeout(() => resolve(x), 2000);
    });
}

async function f1() {
    let x = await resolveAfter2Seconds(10);
    console.log(x); // 10
}

f1()
```

await: Warten bis promise resolved ist

await und then Async Songs Demo

await

```
import dataService from "./promiseDataService.js";

// view refs

function renderSongs(songs) { /* … */ }

// Controller
async function getAndRenderSongs() {
  songslist.innerHTML = "Loading Songs";
  const songs = await dataService.promiseSongs();
  renderSongs(songs);
}

btnElement.addEventListener('click',
  getAndRenderSongs);
```

.then

```
import dataService from "./promiseDataService.js";

// view refs

function renderSongs(songs) { /* … */ }

// Controller
function getAndRenderSongs() {
  songslist.innerHTML = "Loading Songs";
  dataService.promiseSongs()
    .then(songs => renderSongs(songs));
}

btnElement.addEventListener('click',
  getAndRenderSongs);
```

Fetch

Doppeltes then oder await

fetch Request (then)

```
function fetchAndThenLog(fileURL) {
  fetch(fileURL)
  .then(response => {
    console.log(response)
    return response.text();
  })
  .then(text => console.log(text));
}
```

Einfacher Request (await)

```
async function awaitFetchAndLog(fileUrl) {
  const response = await fetch(fileUrl);
  console.log(response);
  const responseText = await response.text();
  console.log(responseText);
}
```

```
▸ Response {type: "basic", url: "http://localho
  redirected: false, status: 200, ok: true, …}
here is some text
```

Einfacher Request

```
const myTextFetchPromise = fetch('./01b-someText.txt');
```

`myTextFetchPromise`	`-> Promise {<fulfilled>: Response}`
`response = await myTextFetchPromise;`	`-> Promise {type: "basic", ...}`
`p2 = response.text();`	`-> Promise {<pending>}`
`p2`	`-> fulfilled`
`text = await p2;`	`-> "here is some text"`

REST

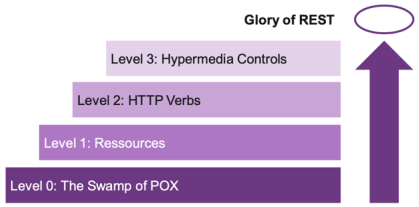
Webservice

„Webservice is a software system designed to support interoperable machine-to-machine interaction over a network“

Ziel

- Dienst (Service) nach aussen zur Verfügung stellen
- Andere Geräte können Webservice über Netzwerk (Web) nutzen

Richardson's Maturity Model



Level 0: The Swamp of POX
Server hat nur einen Endpunkt
→ nur POST

Level 1: Ressources
→ Server hat mehrere Endpunkte
→ Jeder Endpunkt ist unterschiedlicher Request

Level 2: HTTP verbs
GET / POST / DELETE / PUT / PATCH
→ z.B GET um Dinge zu holen und POST um Dinge zu speichern
→ PUT ersetzen einer Ressource
→ GET keine Seiteneffekte
→ POST hat Seiteneffekte

Level 3: Hypermedia Controls
→ Client wird gesagt was mit Ressource gemacht werden muss
→ z.B. Buchung eines Termins ist über andere Ressource möglich

Resource-oriented Architecture

- Ressource
  Alles was wichtig genug ist um eigenständig referenziert zu werden
- Name
  Eindeutige ID der Ressource (z.B. URI)
- Repräsentation
  Ressourcen haben mehrere Repräsentationen
- Links
  Hyperlinks verknüpfen Ressourcen
- Interface
  Uniform Interface
  Benutze Standard `http`-Verben
- Statuslos Kommunikation

Ressource Name

- Ressourcen mit URI identifizieren
- Jede Ressource hat eindeutigen URI
  `orders/1`
  `books/0-330-25864-8`
- Sub-Ressourcen
  Wenn Sub-Ressource ohne Parent nicht existieren kann (**Komposition**)
  `customers/1/orders`
  `topics/1/comments`

Hypermedia As The Engine Of Application State (HATEOAS)

- Prozessgedanke in Ressource
- Media-Typen beschreiben Ressource
- Aktionen werden ausgeführt beim Folgen von Links
- Jede Antwort beinhaltet Application State
- Selbstbeschreibende APIs erzeugen Flexibilität

REST vs. Remote Procedure Call (RPC)

- REST stellt Ressource in den Mittelpunkt
- RPC stellt einzelne Operationen in den Vordergrund

POST example

```
const postsRESTServerURL = 'http://localhost:3000/';
const postsRoute = 'posts';
const commentsRoute = 'comments';

async function getJson(url) {
  const response = await fetch(url);
  return response.json();
}
```

GET-Request

```
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();
}
```

POST-Request

AJAX

Asynchronous JavaScript and XML
with AJAX, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behaviour of the existing page.

- Asynchrone Anfrage and Server (blockiert nicht Event-Loop)
- Inkrementelle Updates statt kompletter Reload
- Keine Seitenwechsel (Single Page App)
- Datenformat heute meist JSON (statt XML)

Vorteile

- Flüssigere Interaktion
- Seite ohne Neuladen aktualisieren
- Kürzere Zeit bis zum First Meaningful Point, da Daten nach Bedarf geladen werden
- In Formularen wird Cursor nicht aus aktuellen Feld genommen

Nachteile bei Single-Page Apps

- Zurück-Funktion und URLs müssen separat programmiert werden (History-API)
- Darstellung und Verwaltung von Loading Indicators

Mehrfachversuche

```
function resilientGetJSON (url, processJSON, refetchTries = DEFAULT_REFETCH_TRIES) {
  const controller = new AbortController();
  setTimeout(() => controller.abort(), FETCH_ABORT_AFTER_MSEC);
  const fetchInitOptions = {signal: controller.signal};
  fetch(url, fetchInitOptions)
    .then(response => {
      if (response.ok) {
        return response.json();
      } else {
        return Promise.reject();
      }
    })
    .then(res => processJSON(res)).catch(() => {
      if (refetchTries > 0) {
        resilientGetJSON(url, processJSON, refetchTries - 1);
      } else {
        processJSON({}); //fail
      }
    });
};
```

Timeout für abort

Response != ok => reject

Catch: Nochmals probieren

Abortable Fetch

```
async function fetchWithTimeout(resource, options = {}) {
    const timeout = 8000;
    const controller = new AbortController();
    const id = setTimeout(() => controller.abort(), timeout);
    try {
        const response = await fetch(resource, {signal:
              controller.signal});
        if (response.status === 200) {
            return response.json();
        } else {
            return response;
        }
    } catch (error) {
        throw error;
    } finally {
        clearTimeout(id);
    }
}
```

Polling

```
async function updateChatFromServer () {
    messages = await dataService.getChat(chatId); updateView();
}
async function joinChat (event) {
    event.preventDefault();
    try {
        joinError = false;
        messages = await dataService.addToChat(chatId, `${handle}
              joined chat`); state = STATE_CHATTING;
        setInterval(updateChatFromServer, 10000);
    } catch {
        joinError = true;
    }
    updateView();
}
joinChatForm.addEventListener('submit', joinChat);
```

Clean Code

Namen Beispiel

- Schlechtester Name für Variable?
  data
- Zweitschlechtester Name?
  data2
- Drittschlechtester Name?
  data_2

Namensgebung

- Funktionen → Verben
- Klassen → Substantive (Nomen)

„A name must be *short, intuitive, descriptive*“

```
const a = 5;   vs.   const postCount = 5;
```

Namen sollten **aussprechbar** sein

```
const cDOB = 0;   vs.   const clientDateOfBirth = 0;
```

- Bekannte Wörter
- Spezifische Namen
- Konsistenz (Project Glossary)
- Kontext muss nicht wiederholt werden
- Namen auf passender Abstraktionsebene

Funktionen

Funktionen sollten…
… **kurz** sein
… **eine** Aufgabe erledigen
… **möglichst wenig Argumente** haben
… einen **beschreibenden Namen** haben und **nur das tun**

AirBnB Styleguide Regeln

`no-var`	var nicht verwenden
`prefer-const`	let nur für Variablen mit dynamischer Zuordnung
`quotes`	Strings immer mit single-quotes
`no-param-reassign`	Parameter nicht in Funktion ändern (auch nicht Parameter-Properties)
`operator-linebreak`	Kein Linebreak direkt vor oder nach Zuweisung (=)
`eqeqeq`	=== und !== nutzen (nicht ==, !=)
`space-infix-ops`	nicht const x = y + 5
`eol-last`	Am Ende des Files ein Return
`no-multiple-empty-lines`	nur 1 Leerzeile
`semi`	Zeilen sollten mit Semikolon abgeschlossen werden

Linting

- Statische Quellcodeanalyse
- Prüfung gegen hinterlegten Regelsatz
- Unterstützt bei Behebung von Regelverstössen
- Verbessert Codequalität & Codierungsstil

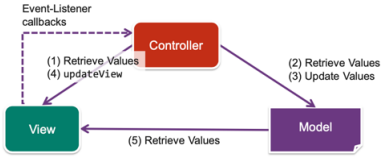
While-Schleifen nicht verwenden

```
'@web-and-design/wed/no-while': 'warn',
for (var i = 0; i < xs.length; i++) {
console.log(xs[i]);
}
xs.forEach((x, i) => console.log(x))
```

Kein switch/if else für Aktionen

```
'@web-and-design/wed/use-action-map': 'warn',
const evalLookup = {scissors: { scissors: 0, stone:
1, paper: -1, }, ...};
evalLookup[playerHand][systemHand];
function getGameEval(playerHand, systemHand) {
 return evalLookup[playerHand][systemHand];
}
```

MVC Model-View-Controller



- Model
  Datenmodell
  App Zustand
  User Daten
  Logik
  Keine View Refs
- View
  Anzeige
- Controller
  User Eingaben
  View-Wechsel (Routing)

JavaScript

```
<!-- type=module -> ESM wird aktiviert -->
<head>
    <title>Test</title>
    <script src="index.js" type="module"></script>
</head>
```

Typeof

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

Falsy / Thruthy

false-Werte	**true-Werte**
false	"0"(String)
0 (zero)	"false"(string)
(empty string)	[]
null	{}
undefined	
NaN	

Number

- Nach Definition sind alle Zahlen "floats"(Gleitkommazahlen)
- Engines versuchen floats auf integers abzubilden - falls möglich

NaN ("Not a Number")

- Ist ein Error-Wert
- 0/0 => NaN
- Hat auch den Type "number"
- NaN == NaN ist immer false
- isNaN() zum überprüfen

Infinity

- Unendlich
- Kann auch negativ sein

Jeder Wert kann in eine Zahl verwandelt werden

- +(true) == 1
- Number(true) == 1
- Number(null) == 0
- Number("abc") => NaN
- Ausnahme: Symbol

parseInt(" string ") | parseFloat(" string ")

- Parst bis zum ersten Fehler

+ - * / Regeln

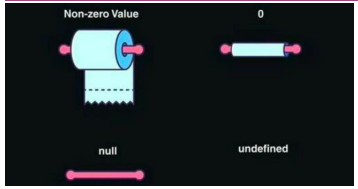
- Punkt vor Strich
- Von Links nach Rechts aufgelöst
- Spezialfälle
  String + Value = String
  Value + String = String
- Ansonsten
  Value (Numerischer Operator) Value = Number

Abstract Equality Comparison Algorithm (==) (===)

```
console.log([] == false);       //true
console.log("" == false);       //true
console.log(null == false);     //false
console.log(0 == "0");          //true
console.log(null == undefined); //true
console.log([1,2] == "1,2");    //true
console.log(NaN == NaN);        //false
console.log([] == ![]);         //true


console.log(false === false);   //true
console.log(4 === 4);           //true
console.log(false === false);   //true
console.log([] === false);      //false
console.log("" === false);      //false
console.log(null === false);    //false
console.log(0 === "0");         //false
console.log(null === undefined);//false
console.log([1,2] === "1,2");   //false
console.log(NaN === NaN);       //false
console.log([] === ![]);        //false
```

null / undefined



Array

- Keine fixe Länge
- Index beginnt bei 0
- Array bietet Iterator-Methoden an

.filter()

```
const words = ['spray', 'limit', 'elite', 'exuberant',
      'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]
```

.entries()

```
const array1 = ['a', 'b', 'c'];
const iterator1 = array1.entries();
console.log(iterator1.next().value);
// Expected output: Array [0, "a"]
console.log(iterator1.next().value);
// Expected output: Array [1, "b"]
```

.concat()

```
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);
console.log(array3);
// Expected output: Array ["a", "b", "c", "d", "e", "f"]
```

.find()

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find(element => element > 10);
console.log(found);
// Expected output: 12
```

.findIndex()

```
const array1 = [5, 12, 8, 130, 44];
const isLargeNumber = (element) => element > 13;
console.log(array1.findIndex(isLargeNumber));
// Expected output: 3
```

.forEach()

```
const array1 = ['a', 'b', 'c'];
array1.forEach(element => console.log(element));
// Expected output: "a"
// Expected output: "b"
// Expected output: "c"
```

.reduce()

```
const array1 = [1, 2, 3, 4];
// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);
console.log(sumWithInitial);
// Expected output: 10
```

.map()

```
const array1 = [1, 4, 9, 16];
// Pass a function to map
const map1 = array1.map(x => x * 2);
console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

.some()

```
const array = [1, 2, 3, 4, 5];
// Checks whether an element is even
const even = (element) => element % 2 === 0;
console.log(array.some(even));
// Expected output: true
```

.every()

```
const isBelowThreshold = (currentValue) => currentValue < 40;
const array1 = [1, 30, 39, 29, 10, 13];
console.log(array1.every(isBelowThreshold));
// Expected output: true
```

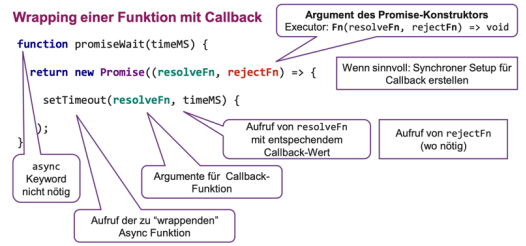
.flat()

```
const arr1 = [0, 1, 2, [3, 4]];
console.log(arr1.flat());
// Expected output: Array [0, 1, 2, 3, 4]
const arr2 = [0, 1, 2, [[[3, 4]]]];
console.log(arr2.flat(2));
// Expected output: Array [0, 1, 2, Array [3, 4]]
```

.flatMap()

```
const arr1 = [1, 2, [3], [4, 5], 6, []];
const flattened = arr1.flatMap(num => num);
console.log(flattened);
// Expected output: Array [1, 2, 3, 4, 5, 6]
```

Callback



Funktionen

- Funktionen sind „First-Class Citizen“
  können in Variablen gespeichert werden
  können als Parameter übergeben werden
  können von Funktionen zurückgegeben werden

- Besitzen offene Parameter-Liste
  - mehr oder weniger als deklarierte Anzahl Parameter übergeben werden
  - alle Parameter werden in `arguments` abgelegt
- Funktionen besitzen Properties
- Normale Funktion erzeugt eigenen Scope

Als Parameter und Rückgabe

```
function add(a, b){
    return a + b;
}
function minus(a, b){
    return a - b;
}
function calc(fn, a ,b){
    console.log(fn(a,b));
}

calc(add, 3, 4);
calc(minus, 3, 4);
```

```
function addTo(a){
    return function(b){
        return a + b;
    }
}
const addTo3 = addTo(3);
addTo3(4);
```

Offene Parameter-Liste

- arguments beinhaltet alle Parameter, welche übergeben wurden
- arguments ist kein Array
  - → `Array.from(arguments)`

Properties

- Funktionen besitzen Properties
- .name beinhaltet den Namen der Funktion
  - Anonyme Methoden besitzen keinen "name"
  - Dieser Name wird für den Stacktrace genutzt
  - Falls kein Name angegeben, loggen moderne Browser den Variablenamen
- .length beinhaltet Anzahl Parameter der Funktion

Rest-Parameters

- Mit `...name` kann man letzten Parameter als „Rest-Parameter“ definieren
- Parameter wird mit restlichen Werten gefüllt
- keine restlichen Parameter erzeugt leeres Array

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

Overloading

- JavaScript kennt kein „Function Overloading“
- Bei gleichen Funktionsnamen überschreibt die zuletzt definierte die vorhergehenden
- Lösung: Interne Wiche (typeof & arguments) oder sinnvolle default-Werte definieren

```
jQuery.fn.init = function( selector, context ) {
    //...
    if ( !selector ) { return this;
    }
        // Handle HTML strings
    if ( typeof selector === "string" ) {
        //...
    } else if ( selector.nodeType ) {
        //...
    } else if ( jQuery.isFunction( selector ) ) {
        //...
    }
    //...
    return jQuery.makeArray( selector, this );
};
```

Scope

- Jede Funktion und Objekt generiert einen neuen Scope
- Innerhalb von einem Scope kann man auf
  - dessen Variablen
  - gloable Variablen
  - Variablen aller „Parent“-Scopes zugreifen (Closure, Werte bleiben erhalten)
- Ein script-Tag erzeugt keinen Scope
- ECMAScript Module erzeugen einen Scope
- Global Namespace Pollution
  - Zu viele Variablen und Funktionen im globalen Scope
  - Variablen oder Funktionen mit identischem Namen überschreiben sich

```
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
```

Strict Mode / 'use strict'

- Erste Linie in File / Funktion
- Strict Mode wird vererbt
- Eliminiert „fails silently“
- Falls eine Variable ohne var definiert wird
- Falls Read-only werte gesetzt werden (ohne strict: ignoriert)
- Eliminiert Probleme wechen es Compiler verunmöglicht Code zu optimieren
- Security wird „leicht“ verbessert
- Wichtig: Das Laufzeitverhalten ändert sich!
- Wichtig: Strict Mode kann nicht mehr entfernt werden
- Wichtig: Strict Mode muss pro File aktiviert werden
- Wichtig: Strict Mode sollte nicht mit Non Strict Mode zu einem File vereint werden
- Immer verwenden ausser Legacy Code, Files werden zusammengefügt
- Klasse und ESM sind immer Strict-Mode

```
function a() {
    a1 = 1; //ok
}

function c() {
    'use strict';

    function b() {
        b1 = 1; //error
    }
    b();
    a();
}
c();
```

Arrow-Funktionen

- auch Lambda genannt
- Sinnvoll für (sehr) kleine Funktionen z.B. als Filter Parameter
- Bei Arrow Funktionen ist Context immer auf das selbe Objekt gebunden
-

```
[ 'a', 'b', 'c' ].forEach((elem, index) => console.log(index
      +":"+elem));
const array = [1,2,3,4].map(x => x*x);
const filteredArray = array.filter( elem => elem > 5);
console.log(array.every( elem => elem > 5));
console.log(((()=>{ const x = 9;
    const y = 11;
    return x + y;
})());
```

Classes

- instanceof
  - Überprüft ob Objekt Instanz einer Klasse ist
- super
  - Zugriff auf Parent
- private verwendet prefix #

```
class AlarmClock extends Clock { // define class
    #timer // private field
    currentTime // field (optional)

    constructor(currenTime = new Date()) {
        super(); // super call
        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;
    }
}
const clock = new AlarmClock(); // create instance
console.log(clock instanceof AlarmClock); // true
console.log(clock instanceof Clock); // true
```

Modules

- Module kann Funktionalität und Werte anderen Modulen zur Verfügung stellen
- kann von anderen Module exportierte Funktionalität und Werte nutzen
- sind immer 'use strict'
- werden in der richtigen Reihenfolge geladen
- Es gibt andere Varianten um „Global namespace pollution“ zu lösen
- Im Node.js wird aktuell oft ein anderer Syntax verwendet
- Um ESM im Node.js nutzen zu können ist Endung *.mjs nötig

Exporting

Named exports

```
export function f() {}
export const one = 1;
export {foo, b as bar};
```

Default exports

```
export default function f() {}
// Replacement for 'const' (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
```

Importing

Named imports

```
import {foo, bar as b} from
      './some-module.js';
```

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 (modules with side effects)

```
import './some-module.mjs';
```

Spread {...}

```
const arr = [1, 2, 3];
const arr2 = [...arr]; // like arr.slice()

arr2.push(4);
//  arr2 becomes [1, 2, 3, 4]
//  arr remains unaffected
```

```
const obj1 = { foo: 'bar', x: 42 };
const obj2 = { foo: 'baz', y: 13 };

const clonedObj = { ...obj1 };
// { foo: "bar", x: 42 }

const mergedObj = { ...obj1, ...obj2 };
// { foo: "baz", x: 42, y: 13 }
```

Destructing

```
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

- Implementieren API's die noch nicht zur Verfügung stehen
- Fixen JavaScript Bugs in alten Browsern
- Polyfill können wieder entfernt werden, falls nicht mehr benötigt