Webseite/Webapplikation: Beinhaltet die Logik der Anwendung Web-Server: Nimmt Anfragen vom Netzwerk entgegen und leitet diese an die entsprechende Webseite weiter. Server. Stellt die Hardware und das OS für den Weh-Server hereit

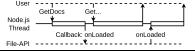
Eine JavaScript Runtime mit einem riesigen Package Ecosystem Vorteile: Serve und Client in selber Sprache (JavaScript), nativer Support von JSON, einfaches und schnelles Deployment, modula rer Aufbau mit zahlreichen Packages, wenig "Magie

- Node is baut auf Chrome's V8 JavaScript Engine auf.
- ⇒ Es ist aber kein Web-Framework und keine Programmiersprache

Event-Driven, Non-Blocking I/O-Model

Node.js verwendet nur einen Thread. Wird ein Event ausgelöst (z.R. GetDocs) nimmt Node dieses entgegen übergibt es an eine API (z.B. File-API), registriert ein Callback-Event und arbeitet dann weiter. Wird das Callback-Event ausgelöst, wiederholt Node diesen Prozess bis zum Ende.

⇒ Sinnvoll für viele kleine, delegierbare Aufgaben in Single-Threaded Systemen.
⇒ Ab Node js 12 sind jedoch auch "Worker Threads" möglich.



Callbacks und Events

Callbacks: Funktionen, die erst später aufgerufen werden Events: Ereignisse, auf die man sich registrieren kann.

```
fs.readFile('demo.txt', (err, data) => { /* Callback...
button.addEventListener('click', (event) => { /* Event.
```

Callbacks sind 1:1 Verbindungen, Events sind 1:n Verbindungen

Promises

Sind eine Alternative zu Callbacks und lösen die Callback-Hell

```
Variante 1
```

```
try { const data = await fs.promises.readFile('demo.txt') }
catch (err) { /* ... */ }
```

⇒ Der aktuelle Standard in Node is sind immer noch Callbacks.
⇒ Viele Module bieten aber eine Promise-Variante zusätzlich an (zB. fs. promises)

Code-Pakete, die Funktionalitäten für andere Module bereitstel len. Node verwendet den Package Manager npm und kennt zwei Modulsysteme: CommonJS: Standard seit Beginn (module.js / mo dule.cjs). ESM: Verwendbar seit ES2015/Node.js 14 (module.mjs).

```
function funcStuff() { /* .
const valueStuff = '...';
 odule.exports = { funcStuff, valueStuff }:
const module = require('./module.js')
module.funcStuff(); module.valueStuff
                                                                              CommonJS
export default function funcStuff() { /*
import funcStuff, { valueStuff } from './module.mjs';
funcStuff(); valueStuff;
```

Auflösungsreihenfolge: 1. Core-Module ('fs'). 2. Pfade ('./module.mjs'). Suche relativ zum aktuellen Pfad. 3. Filenamen ('module.mjs'). le'). Suche in node_modules und danach bis zum File-System-Root.

→ Module werden vomSystemnur einmal geladen. → Core-Module: http, url, fs, net, crypto, dns, util, path, os, etc.

nackage ison: Beinhaltet die Projektinformationen (Name Scrints Packages etc.) und ist zwingend nackage-lock ison: Beinhaltet die exakten Package-Abhängigkeiten. Wird automatisch aktualisiert und gehört ins Repo. Native Module: Beinhalten nativen Code. NVM: Versionsmanager für Node.js

Was ist Express.is?

Bekanntes Web-Framework für Node. Baut auf dem MVC-Pattern auf und verwendet Middlewares für die Request-Bearbeitung.

Model-View-Controller (MVC)

Beschreibt ein Frontend Design Pattern, Model: Beinhaltet die Daten und Datenaufbereitung. View: Stellt die Daten dar. Contro ler: Verknüpft das Model mit der View.



Middlewares sind ein Stack von Anweisungen, welche für einen Request ausgeführt werden. Die Reihenfolge der Registrierung nestimmt die Ausführungsreihenfolge

Express is stellt ab V4 viele Middlewares zur Verfügung (zuvor "Connect"-Plugin).

Authentisierung

Cookies/Sessions

Cookies: Repräsentieren ein kleines Stück von Daten. Der Server schreibt die Cookies zum Client, der Client sendet alle seine Cookies hei jedem Request mit Sessions: Bauen auf Cookies auf. Beim ersten Request wird eine Session-ID vom Server erstellt und als Cookie zum Client gesendet. Der Server kann den Benutzer nun bei jedem Request mit dieser ID authentifizieren.

Session/Cookies sind nicht Stateless, Tokens hingegen schon.
 Authentisierung: Wer bin ich? (Pin, 2FA etc.) Autorisierung: Was darf ich?

Tokens

Tokens: Beinhalten alle Informationen für die Authentisierung und Autorisierung eines Benutzers (Ausstelldatum, Ablaufdatum Benutzerdaten, etc.). Werden vom Server ausgestellt und vom Client bei jedem Request mitgesendet. Vorteile: Sind Stateless und Serverübergreifend (vgl. Git-Tokens) Nachteile: Können geklaut werden (Lösung: kurzes Ablaufdatum und Invalidierung)

JSON Web Tokens (JWT): Offener Standard für die Erstellung von Tokens. Wird über den HTTP-Header mitgesendet und beinhaltet Header (Algorithmen, etc.), Payload (Daten) und Signatur.

Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoiSm9obi BEb2UifQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

⇒ Einige Services bieten ihre Tokens öffentlich an (zB. REST API Tokens). Tokens nur über sichere Verbindungen senden.

Weiteres

Routing: Zuordnung von Routes (GET /orders) zu Actions (ordercon troller.getAll). Template Engines: Engines für das Definieren und Rendern von HTML-Templates über einen einfachen Syntax AJAX: Asynchrones Laden von Webinhalten (Filter Single Page Applications, etc.). Schwierigkeiten bei SEO und Undo/Redo. Verwendung via fetch('/page').then(res => res.ison()).then(...)

Verwendung von Express.js

const router = express.Router();

router.get('/books/:id/', controller.getBooks)

}
export default customLogSessionMiddleware;

```
import express from 'express';
import bodyParser from "express";
import session from 'express-session
import path from "path";
 import exphbs from 'express-handlehars'
 import customLogSessionMiddleware from './custom-middleware.js'; import router from "./routes.js";
 const app = express();
app.use(express.static(path.resolve('public'))); // Static files app.use(session({ secret: "1234567', resave: false, savelninitialized: true })); app.use(bodyParser.urlencoded({ extended: false })); // req.body
 app.use(customLogSessionMiddleware);
app.use(router);
 const hbs = exphbs.create({ extname: '.hbs' });
hbs.handlebars.registerHelper('concat', (a, b) => a + '-' + b);
app.engine('hbs', hbs.engine);
 app.set('view engine', 'hbs');
app.listen(3001, '127.0.0.1', () => {
    console.log('Example app listening on port 3001!');
                                                                                                            app.js
import express from "express";
import { controller } from './controller.js'
```

```
.get(controller.getOrders)
.post(controller.postOrders) // Multiple callbacks possible
router.all('/*', controller.default) // Uses pattern matching
export default router:
                                                                         routes.is
     default(req, res) { res.render('index', { id: 0 }); }
     getBooks(req, res) { res.render('index', {id: req.params.id
books: [{ name: 'A'}] }); }
    getOrders(req, res) { /* ... */ } postOrders(req, res) { /* ... */ }
```

```
export const controller = new Controller():
                                                     controller.js
function customLogSessionMiddleware(req, res, next) {
   console.log(req.session.counter)
  req.session.counter += 1;
   next(); /* Calls next middleware */
```

```
<head> <meta charset="utf-8"> <title>App</title> </head>
<body> {{{body}}} </body>
```

```
<h1>{{id}}</h1>
{{concat "book" id}}
 {{/each}}
                               views/indexhbs
```

NeDB ist eine NoSOI -Datenbank Alle Daten werden in JSON-Dokumenten abgespeichert. Relationen müssen manuell gesetzt und verwaltet werden (z.B. via doc. id).

```
import Datastore from 'nedb'
const db = new Datastore({ filename: './books.db',
db.insert({ name: 'A' }, (err, doc) => { /* - db.find({ name: 'A' }, (err, doc) => { /* - db.find0e({ name: 'A' }, (err, doc) => { /* - db.update({ name: 'A' }, (err, doc) => { /* - db.update({ name: 'A' }, { name: 'B' }, { }, (err, num) => { /* - /* });
                                                                                                                                                                             data-store.is
```

Was ist TypeScript?

TypeScript ist eine Programmiersprache, welche JavaScript mit Typen und weiteren Syntaxelementen ergänzt. Ein Pre-Processor übersetzt TypeScript in JavaScript, d.h. es existiert kein Runtime Typechecking. Jedes valide JS ist valides TS.

- Die Typen müssen oft mitinstalliert werden (npm i -D @types/node)
- Statt Node is kann auch ts-node verwendet werden (bietet .IITCompilation von TS).

Strict Mode

nolmplicitAny: Keine untypisierten Variablen. alwaysStrict: Verwendet automatisch ein "use strict" im JS-File. strictNull-Checks: null und undefined sind nicht mehr Teil der Basistynen (explizite Deklaration nötig). strictFunctionTypes: Strenge Überprüfung von Funktionstypen, strictPropertyInitialization; Klassen eigenschaften müssen initialisiert werden.

⇒ Weitere Einstellungen sind nolmplicitThis und strictBindCallApply

Spezifikation Basistypen

boolean, number, string, null; Wie in anderen Sprachen, undefined: Variable deklariert aber kein Wert zugewiesen anv Beliebiger Wert, Zuweisung in beide Richtungen beliebig möglich, unknown: Unbekannter Typ. Zuweisung zu unknown beliebig möglich. Zuwei sung von unknown erst nach Type Narrowing. Type Inference: Ohne Typdeklaration wird der Typ automatisch bestimmt.

```
let aAny: any = true
aNumber = aString; /* NOK */ aAny = aString; /* OK *, aNumber = aAny: /* OK */ aUndefined = undefined: /* OK *,
aString = undefined; /* NOK (in Strict Mode) */
```

Komplexe Typen und Typdeklaration

Arrays, Tupels und Enums wie in anderen Sprachen. Union Types: Zusammengesetzte Typen (boolean | string). String/Number Literal Union Types: Zusammengesetze Typen aus Text-oder Zahlenwerten, ähnlich wie Enums. Wichtig: Keine Type Inference bei Tupeln! Diese werden als "Union Type Arrays" erkannt.

Neue Typen können mit type CustomType = ... deklariert werden.

```
enum EnumType { A, B, C }
type StrLitUnionType = 'A' | 'B' | 'C'
 let aUnionType: number | undefined
 let anArray: number[] = [1, 2, 3]
let anUnionArray = [1, 'abc'] // Inferred (number|string)[]
 let aTupel: [number, string] = [1, 'abc']
let anEnum = EnumType.A
```

Type Narrowing

TypeScript verwendet Flussanalyse, um die tatsächlichen Typen von Variablen zu bestimmen. Zuweisungen werden so möglich Achtung: unknown braucht explizites Type Narrowing mit typeof

```
let aNumberOrString: string | number
let aUnknown: unknown:
let aNumber: number
aNumber = aNumberOrString /* NOK */
aNumber = aUnknown
aNumberOrString = 1
aUnknown = 1
aNumber = aNumberOrString
aNumber = aUnknown
```

Funktionen

custommiddleware.is

Wie in anderen Sprachen, Erlauben Defaults und optionale Para meter. Mehrere Signaturen pro Funktion möglich (Function Overloading). Funktionen als Parameter möglich

```
function myFuncA(a: string | number = '', b?: number):
function myFuncB(a: (num: number) => number): void { }
 myFuncA('abc', 1); /* 0K */ myFuncA(); /* 0K */ myFuncA('abc'); /* 0K */ myFuncB((num) => num + 1); /* 0K *
```

Klassen und Interfaces

Wie in anderen Sprachen. Properties lassen sich im Konstruktor definieren (automatische Generierung und Initialisierung). Gene rics sind möglich. Mit Readonly<T> können alle direkten Felder ei ner Klasse Tunveränderlich gemacht werden

Type Assertions: Erlauben Spezialisierung und Generalisierung eines Typs. Im Gegensatz zu Type Casting sind inkompatible Typen nicht erlaubt. Structural Typing: JS-Objekte können, wenn sie vom Typ her passen, an TS-Objekte zugewiesen werden.

```
Structural Typing verwendet natives "Duck-Typing" aus JavaScript
class AClass {
```

```
#val1: numbe
  private val2: string // Private: TS Default readonly val3?: number // Public, Optional, Readonly val4 = 1 // Public, Static, Readonly
 constructor(val1: number, val2: string, val3?: number) {
   this.#val1 = val1; this.val2 = val2; this.val3 = val3;
 set val1(val: number) { this.#val1 = val } // ES6
get val1() { return this.#val1 } // ES6
interface AInterface<T> {
    readonly valA: T // Public (Never static, private,
    valB?: boolean // Public (Never static, private,
    func(): void // Public (Never static, private,
class ASubClass<T> extends AClass implements AInterface<T> {
    constructor(public valA: T, private valC: number) {
         func() { /* ... */ }
let aClass = new AClass(1, 'abc', 2);
let aSubClass = new ASubClass<number>(1, 2);
let aIntA: AInterface<number> = { valA: 1, func() {} } // OK
let aIntB: AInterface<number> = { valZ: 'abc' } // NOF
// Iype Assertions
let aTypeA = aSubClass as AClass; // OK
let aTypeB = aSubClass as number; // NOK
```

Globale Variablen aus nicht TS-Files können mit declare let aGlobal: ... deklariert werden. Keyof und Template Literal Types er lauben die Generierung von speziellen String Literal Union Types

```
type Keys = keyof { x: any, y: any } // type Keys = 'x' | 'y
type TempLit = `my-${Keys} // type TempLit = 'my-x' | 'my-y
```

Flexible Layout: Verfügbarer Platz wird durch Verbreiterung der Elemente ausgefüllt. Responsive Layout: Verfügbarer Platz wird durch Umordnung/Einblendung von Elementen ausgefüllt (Media-Oueries), Graceful Degradation: Fokus auf neue Browsers, Alle Features nutzen und dann Rückwärtskompatibel machen (Polyfills) Progressive Enhancement: Fokus auf alte Browser. Mit Grundfunktionen starten und dann neue Features zusätzlich anbieten.

Mobile-First: Fokus auf Mobile statt Desktop (z.B. Mobile-Base-CSS).

Im Normalfall werden Flexible Layouts und Responsive Layouts miteinander kombiniert Wichtig, da es zahlreiche Geräte mit unterschiedlichen Screen-Sizes gibt.

Normalize & Reset

Alle Browser haben ein etwas anderes Default-Styling. Reset: Ent femt die meisten Default-Styles. Entwickler muss Konsistenz schaffen. Normalize: Entfernt alle inkonsistenten Default-Styles

Media-Queries

Erlauben spezifisches CSS für unterschiedliche Medien

Typen: screen, print. Dimensionen: width, min-width, max-width, height, min-height, max-height. Zustände: orientation: landscape, ho ver:hover|none, pointer:fine|coarse|none. Operatoren: and, not, only, (Komma: "OR"). Beachte: AND bindet stärker als OR.

Featureabfrage: @supports (display: grid) {...}.

```
body { color: blue; } /* Blue is the Standard @media screen and (min-width: 480px) {
     body { color: green; } /* Green if width > 480px */
@media (hover: none) and (pointer: coarse) {
    body { color: red; } /* Red if Touchscreen */
```

>480px/30em: Mobile. >768px/48em: Tablet. >992px/62em: Laptop. Danach Desktop. Das Verhältnis von CSS-Pixeln (px) zu echten Pixeln ist abhängig vom Zoom/Gerät.

Viewport & Link

Media-Queries können im link media=...>-Tag gesetzt werden. Dadurch wird nur das benötigte CSS geladen (Ladezeitoptimierung).

Mit <meta name="viewport" content="width=device-width,initialscale=1"> wird das "intelligente Zoomen" von Mohile-Browsem unterbunden. Wichtig damit die Media-Queries greifen.

CSS-Features

Boxmodell: box-sizing: content-box bedeutet, Boxgrösse ist width -2*padding + 2*margin und Inhalt ist width. border-box bedeutet, Boxgrösse ist width und Inhalt ist width - 2*padding - 2*margin.

Element	L/R Margin	T/B Margin	Width/ Height	Causes Line-Break	Allows Scroll	Adds Whi- te-Space
inline	Х					х
block	X	×	×	×	х	
inline- block	×	×	×		×	×

L/R: Left/Right, T/B: Top/Bottom, Margin beinhaltet auch Padding

Viewarösse (vh/vw): Prozentual zur Viewarösse, 100vh ist nicht immer korrekt (z.B. sichtbare Browserleiste). Besser 100dvh verwenden.

Prozent (%): Bezieht sich immer auf den Parent, ausser bei trans late(x,y). Funktionen: calc(100vh-5px), min(..., ...), max(..., ...), clamp(min_ideal_may) Scrollen: overflow:visible!bidden!scroll! auto. Scrollbar soll erkennbar sein. Position: position: absolute ist relativ zum ersten Parent mit absolute|relative. fixed ist relativ zum Browserfenster. sticky ist wie absolute, bleibt aber oben/unten am Bildschirmrand haften. relative ist relativ zur aktuellen (normalen) Position. static ist normal im Fluss.

Custom Properties

Erlauben Variablendefinition in CSS. Werden bei der ersten Ver wendung im DOM (Root-to-Leaf) ausgerechnet (Aufpassen bei "berechneten" Werten). Können mit style getPropertyvalue(...)/ style.setProperty(...) ausgelesen und verändert werden.



Container/Parent

Aktivierung mit display: flex|inline-flex. Gilt für alle direkten Kinder. inline-flex ist nach innen flex, nach aussen inline.

Hauptachsendefinition: flex-direction: rowlcolumn lrow-reverse Lco lumn-reverse. Umbruchverhalten: flex-wrap:nowrap|wrap. Hauptachsen-Ausrichtung (pro Zeile): justify-content:flex-start|flex end|center|space-around|space-between|space-evenly. Querachsen-Ausrichtung (pro Zeile): align-items:stretch|flex-start|flexend|center|baseline. Querachsen-Ausrichtung (Gesamtblock): align-content:[justify-content]|stretch.

- flev-wran heachtet flev-hasis width height undignoriert flev-shrink flex-wrap:wrap garantiert keine exakte "Grid"-Ausrichtung der Elemente.
- Achtung: row bedeutet zB. auf Japanisch von oben/unten und rechts/links

Item/Child

Verhält sich wie inline-block. Werden anhand der "Source-Order" im HTML platziert. flex-grow/flex-shrink sind Verhältniswerte flex-basis bestimmt die Standardgrösse.

Wachstumsverhalten: flex: [flex-grow] [flex-shrink] [flex-basis] (Standard: 0 1 auto). Umsortierung: order: [number] (Standard:). Querachsen-Ausrichtung: align-self: [align-items].

- flex-basis: auto bedeutet "nimmmeine aktuelle Grösse".
- ⇒ fley-grow: 2 ist 2x so gross wie fley-grow: 1 Rhedeutet kein Wachstum

Container/Parent

Grid

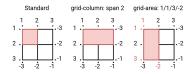
Aktivierung mit display: grid. Gilt für alle direkten Kinder.

Anzahl und Grösse von Zeilen/Snalten: grid-template-co. lumns|grid-template-rows: [measure]. Row-Ausrichtung (pro Zelle): justify-items: start|end|center|stretch. Column-Ausrichtung (pro Zelle): align-items: [justify-items]|baseline.

⇒ Flexbox Eine Primär-Achse, beliebige Anzahl von Elementen, Kinder bestimmen Grösse.
⇒ Grid: Zwei Primär-Achsen. fixe Anzahl von Elementen. Parent bestimmt Grösse.

Item/Child

Lässt sich beliebig anhand der Linien platzieren.



Positionsbestimmung: grid-column-start: x1, grid-column-end: x2, grid-row-start: y_1 , grid-row-end: y_2 , grid-column: y_1/y_2 , grid-row: y_1/y_2 , grid-area: $y_1/x_1/y_2/x_2$. Row-Ausrichtung: justify-self: start|end|center|stretch. Column-Ausrichtung: align-self: [justify-self]. Row+Column-Ausrichtung: place-self: [justify-self].

Inumber 1 fr: Fraktion des verfügharen Platzes Erlauht Dezimal Kann nicht schmaler als das längste Wort werden (Overflow wird vermieden), min-content; Breitenanpassung an das längste Wort. max-content: Breitenanpassung an den gesamten Text. minmax(min, max): Gleichmässige Platzverteilung zwischen Min. und Max. fitcontent: [length]: Entspricht minmax(auto, [length]). repeat([number]|auto-fill|auto-fit, [measure]): Wiederholt Wert so oft wie angegeben, auto: Automatische Grössenanpassung.

Berechnung: Grid mit width: 100px und grid-template-columns: 10px 1fr 2fr. dh 99nx freier Platz dh 99/(2+1)=39nx dh 1fr=39nx und 2fr=69nx

Seit 2021 gilt in der Schweiz für die öffentliche Verwaltung (Bund. SBB. etc.) der eCH-0059 Accessibility Standard (Konformitätsstufe AA)

Wie kann man Accessibility sicherstellen?

Farbenblindheit: Informationen nicht nur mit Farben codieren Stattdessen Mehrfach-Codierung anwenden (Farbe & Form, Farbe & Icon, etc.), Simulation u.a. via Chrome Dev Tools möglich.

Kontrast: Wichtig für Personen mit Sehschwäche oder Alter ü. 50. Kontraststufe 5.7:1 für ü. 50 (AA), 15.9:1 für ü. 80 (AAA). Testing u.a. via Chrome Accessibility Audit oder Wave-Plugin.

Auszeichnung von Medien: Bilder sollten immer einen Alt-Text haben. Art von Bild, Kontext, Inhalt und allenfalls Text als Zitat angeben. (A comic strin of a confused man saving "OK")

Zoombarkeit Zoom nicht unterbinden (kein user-scalable = 0, etc.). Schrift soll via Browsereinstellungen separat skalierbar sein (Verwendung von em/rem/% Statt px).

Animationen: Animationen sollten abstellbar sein, um Ablenkungen (2.B. bei ADHS), Epilepis e und Migräne zu verhindem (@media (prefers-reduced-motion) (3). Animationen von opacity immer in Kombination mit visibility: visible|hidden verwenden.

Bedienbarkeit mit der Tastatur. Alle wichtigen Elemente sind in der richtigen Reihenfolge fokussierbar (Tab). Fokus soll sichtbar sein. Dazu Standard-Controls nutzen und Controls nicht mit CSS umsortieren (kein flex-direction: reverse, etc.).

Screenreader. Für Unterstützung keine Headling-Levels auslassen, semantische Elemente verwenden (anstatt ARIA-Attribute), versteckte Skip-Links ("Zum Hauptinhalt") am Seitenanfang einbauen, Lang-Attribut setzen, Tabellen-Headlings für Zeilen/Spalten verwenden, Captions verwenden, Inputs mit Labels versehen

Custom-Controls: Nicht selber entwickeln. Bestehende Controls mit angepassten Styling verwenden (Star-Rating aus Radio-Buttons). Drag & Drop vermeiden. Gute Control-Libraries verwenden.

Automatische Accessibility Tests sind limitiert. Manuelle Tests immer empfohlen.
□ Top-5 Probleme: Kontrast, kein Alt-Text, leere Links, keine Labels, leere Buttons.

Wer braucht was?

Sehbehindert Kontraste, Zoombarkeit, Trennung von Inhalt und Layout. Höhehindert Audiotranskription, visuelles Feedback, einfache Texte. Blind: Gute Dokumentstruktur, Alt-Texte, Tastaturbedienung, Motorische Einschränkung: Grosse Schaltflächen, Tastaturbedenung, keine ungewollten, automatischen Aktionen. Lem- & Aufmerksamkeitsstörung: Einfache Texte, lesbare Schrift arten, keine ablenkenden Elemente.

⇒ Betrifft 21% der Schweizer Bevölkerung.

Security

OWASP Top 10

A01: Broken Access Control A02: Cryptographic Failures A03: In jection A04: Insecure Design A05: Security Misconfiguration & A06: Vulnerable & Outdated Components A07: Identification & Authentication Failures A08: Software & Data Integrity Failures A09: Security Logging & Monitoring Failures A10: Server-Side Request Forcer

Fokusthemen

A01.1: Insecure Direct Object References (IDOR)

Beschreibung: Eine Webseite ist verwundbar, wenn Daten ohne Sicherheitsmechanismen über direkte Referenzen erreichbar sind. Beispiel: Die Daten eines Nutzers lassen sich über myste.ch/user10 einsehen. Die Berechtigungen des Aufrufers werden aber nicht geprüft. Ein Angreifer kann nun z.B. solange IDs ausprobieren, bis er auf die Seite eines Nutzers kommt.

Lösung: Alle Seiten mit korrekter Berechtigungskontrolle ausstatten. Bei Express js z.B. via Middlewares, welche die req.params.id mit der aktuellen Nutzer-ID vergleichen.

A01.2: Cross-Site Request Forgery (CSRF)

Beschreibung: Eine Webseite ist verwundbar, wenn sie Formulare zusammen mit Cookies verwendet und die Herkunft des Formulars nicht überprüft. Beispiel: Ein Angreifer bringt einen eingelogsten User dazu, auf evit. en zu gehen. Auf dieser Seite wird (automatisch) ein Formular an my-site. Ord/detexecount gesendet. Da der Browser die Cookies der realen Webseiten mitsendet, wird die Aktion im Namen des Nutzers ausoeführt.

Lösung: Formulare mit einem CSRF-Token versehen und beim Request überprüfen. Bei Express.js z.B. das csurf Plugin verwenden. Altemativ keine oder SameSite-Cookies verwenden.

A01.3: Replay Attacks

Beschreibung: Eine Webseite ist verwundbar, wenn Aktionen unbeabsichtigt mehrmals ausgeführt werden Können. Beispiel: Eine Webseite belohnt die Nutzer für jede korrekte Aufgabe mit Punkten. Ein Nutzer kann nun z.B. eine einzige Aufgabe mehrmals absenden, um so mehrere Punkte zu erhalten.

Lösung: Keine "generische" Lösung möglich. In diesem Fall z.B. die Aufgaben abspeichem und nur einmal zählen.

A02: Cryptographic Failures

Beschreibung: Eine Webseite ist verwundbar, wenn sie veraltete oder riskante Kryptoalgorithmen, mangelte Zufallszahlen, hart-kodierte Passwörter oder ähnliches verwendet. Beispiel: Eine Webseite sendet die Logindaten eines Benutzers unverschlüsselt und ohne HTTPs an den Server. Ein Angreifer kann nun z.B. mit einer Network Sniffing Software die Logindaten abhören.

Lösung: Verwendung von HTTPs. Keine sensitiven Informationen in der URL codieren. Externe Auth-Services nutzen.

A03.1: Cross Site Scripting (XSS)

Beschreibung: Eine Webseite ist verwundbar, wenn ein Angreifer seinen Schadcode so auf der Seite abspeichern kann, dass dieser im Browser eines Nutzers ausgeführt wird. Beispiel: Die Eingaben eines Formulars werden ohne "Escaping" an andere Nutzer ausgeliefert. Ein Angreifer kann nun z.B. <a href="mailto:simpeben und so beliebienen Ode ausführen."

Lösung: Escaping/Encoding verwenden. Eingabebereinigung (Sanitizing) via Libraries (XSS, DOMPurify) einbauen. CSP-Header setzen. HTTPOnly-Cookies setzen.

⇒ Für Encoding in HBS {{content}} anstatt {{{content}}} verwenden.

A03.2 Remote Code Execution / Injection

Beschreibung: Ein Server ist verwundbar, wenn ein Angreifer den Server dazu bringen kann, seinen Schadcode (JavaScript,SQL,etc.) auszuführen. Beispiel: Die Eingaben in einem Formular werden mittels eval(_) vom Server in ein bestimmtes Format konvertiert. Ein Angreifer kann nun z.B. while (true), process. exit() (DoS) oder res.end(fs.readdirSync(_).toString()) aus führen.

Lösung: Niemals eval(_), sondern parseInt(_) oder JSON.parse(_) verwenden. Eingabebereinigung einbauen. Rechenintensive Tasks auslagem (gegen DDoS-Attacken). Node.js keine Root-Rechte geben. Globale Scopes und Variablen reduzieren.

⇒ setTimeout(...) und setInterval(...) werhalten sich ähnlich wie eval(...).

A07: Identification & Authentication Failure

Beschreibung: Eine Webseite ist verwundbar, wenn sie u.a. Brute Force-Attacken erlaubt, schlechte Passwörder zulässt oder keine Multi-faktor-Authentisierung verwendet. Beispiel: Ein Angreifer bekommt Zugniff auf ein Nutzerkonto, indem er alle möglichen Passwortkombinationen durchprobiert. Da das Passwort "1234" war, ging die Attacke nur wenige Minuten.

Lösung: Korrekte Authentisierung einbauen. Auth-Service nutzen

Login & Password Handling

Grundsätzlich sollte man, wenn immer möglich, einen externen Auth-Service (z.B. via OAuth) verwenden. Eine Middleware stellt dabei sicher, dass Anfragen auf geschützte Bereiche nur durch autorisierte Benutzer erlaubt sind (z.B. via Token-Validierung).

⇒ Bei grossen Firmen mit eignen OySec-Teams sind eigene Auth-Services in Ordnung.
⇒ Möglichkeiten: OpenID Connect, OAuth, Passwordless (Magic-Links, WebAuthN, TOTP)

Weiteres

CSP-Header. Mechanismus, um das Laden von Ressourcen auf die angegebenen Domains zu beschränken (content-security-Po-Licy: default-src self trusted-service.com). HTTPOINy/SameSite Cookie: Blockiert den Zugriff von Client-Scripts sowie von anderen Webseiten. Secure-Cookie: Begrant Cookies auf HTTPs.

Testino

Testarten

Static: Statische Code-Analyse. Unit Testen einer einzelnen Komponente (Klasse, Modu, dec.). Indegration: Testen von meherren Komponenten. EZE/System: Testen von allen Komponenten. Funktional: Testen von konkreten Anforderungen (Use-Cases). Regression: Testen auf potenzielle Fehler nach Code-Änderungen. Weiteres: Testen von Security, Usability, Performance, Stress, etc.

⇒ Oft wird dabei von einem System Under Test (SUT) gesprochen.

Begriffe & Tools

Test-Gruppe/Fixture: Beinhaltet die Tests. Test-Environment. Umgebung, in der Tests ausgeführt werden. Doubles/Mocks: Fake-Klassen fürs Testen. Phasen: Setup, Exercise, Verify, Teardown

Assertion Libraries: Erlauben eine einfache Spezifikation von Tests (Chal Expect. js). Test-Runners: Führen die Tests aus (Mocha, Cypress, Jest), Mocking Libraries: Erlauben die Erstellung von Mocks (Proxyquire, Sinon. js). DOM-Handling: Erlauben das Testen von Web-Uls (Cypress, Puppeter).

⇒ Achtung: Teilweise wird auch das Test-Environment als "Fixture" bezeichnet.
⇒ Assertion Libraries könnten auch durch throw new Error (...) ersetzt werden.

Prinzipen & Test Smells

Alles Testen, das kaputt ging oder gehen könnte. Neuer Code ist immer "schuldig". Vor einem Push alle Tests laufen lassen. Mocks/Doubles mittels Dependency Injection zulassen.

Smells: Hard-to-Test Code, Production Bugs, Fragile Tests (Stark abhängig von internet togik), Erratic Tests (zufälig Pass/Fail), Developers not Writing Tests, Assertion Roulette (Zuviele Assertions in einem Test), Test Logic in Production Code, Obscure Tests (Zukomplziert), Slow Tests. Test Code Duplication, Conditional Test Logic.

Mocha/Chai

Erlauben Unit-/Integration-Tests. Syntax: expect(_).to.equal(val), .to.be.false, .to.be.a(number), .to.deep.equal(array), .to.throw(Ty-peError), to.not.be.undefined, .to.have.any.key(key), ...

```
jmport chai, (expect) from 'chai'
import chailtitp from 'chai-inttp';
import jsdom from 'jsdom';
import app from ',2mp_js'
chai.use(chaintip),
chai.use(chaintip),
inchai.use(chaintip),
itchai.use(chaintip),
itchaintip),
itch
```

Cypress

Erlaubt komplexe E2E mit einem Mocha/Chai-ähnlichen Syntax. Läuft in einem eigenen Test-Runner mit Browser-Dashboard.

```
describe('Cypress-Fixture', () => {
  it('EZE: Clear button clears the form', () => {
    const input = cy.get('input[name="email"]')
    input.type('Typing...')
    input.type('Typing...')
    cy.get('#clear-button').click()
    input.should('have.value', 'Typing...')
}
```

⇒ cy.url() gibt die aktuelle URL zurück.cy.submit() sendet ein Formular ab.
⇒ cy.contains("Hello") gibt das erste Element mit dem Text "Hello" zurück.

internationalisierun

Begriffe

Internationalisierung (18N): Programmieren auf eine Art, sodass Lokalisierung möglich wird. Lokalisierung (110N) / Globalisierung (611N): Anpassung des Programms (Sprache, Farbe, etc.) an die Sprachregion. Übersetzung (17N): Übersetzung von Texten und Wörtem. Locale: Bezeichnung der Sprachregion (z.B. als String).

⇒ Herausforderungen: Sprache, Wortlänge, Schreibrichtung, Farbbedeutung, etc.

Best Practice

Layout Elementpositionen an kulturelle Benutzergruppe anpassen (z.B. Leserichtung). Textlänge soll Hierarchie nicht beeinflussen (ungünstige Umbrüche). Layout nicht von der Wortsortierung abhängig machen. Eingabefelder. Unterschiede bei Postleitzahl, Telefonnummer, Provinzbezeichnung, etc. beachten. Vollständiger Name anstatt Vor- und Nachname verwenden. Unterschiedliche Datums- und Zahlenförmate zulasser.

⇒ Je nach Region/Zielgruppe ist der Lokalisierungsaufwand grösser oder kleiner.
⇒ Nicht alles lässt sich automatisch lokalisieren (Denke: Farbe, Etiquette, etc.).

Umsetzun

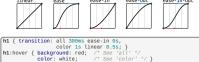
Anführungszeichen: Werden bei <q>_</q> automatisch basierend auf lang-Attribut gesetzt. Standard (JavaScript): ES2021 Internationalisierung. Bibliotheken: FormatJS, Polyglot.js, i18next.

Animationen mit CSS

CSS-Transitions

Erzeugen einen Übergang von Zustand A zu Zustand B. Die Frames dazwischen werden automatisch berechnet (Tweening).

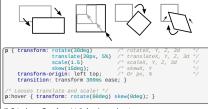
transition-property: CSS-Property zum Animieren. transition-duration: Animationsdauer. transition-timing-function: Beschleuni gungsfunktion. transition-delay: Zeitdauer, bevor die Animation startet. Kurzschreibweise mit transition.



⇒ Alle Animationen werden über eine Ωubic Bezier Ωurve definiert

CSS-Transforms

Verändem die Form/Position eines Elements bei der Anzeige. Kann mit transition animiert werden. Rotationen sind im Uhrzeigersinn. Der Ursprung Kann mit transform-origin: x y festgelegt werden. Bei 30-Transformationen kann perspective: [number] auf dem Parent gesetzt werden (höhere Werte wirken "extremer").



⇒ Beimehreren Transforms ist die Anordnung relevant.
 ⇒ Achtung: Bei Zustandswechsel immer alle gesetzten Transforms wiederholen.

Keyframe Animatione

Frlauhen komplexere Animationen mit mehreren Zuständen

Die meisten Properties mit quantitativen Werten/Farben lassen sich animieren. Nicht animierbar sind display, auto, ur l (...), etc.

- ⇒ Mit der Houdini-API@property) lassen sich auch solche Properties animieren.
- ⇒ JS-Animationen sind grundsätzlich mächtiger, aber nicht so performant.
 ⇒ SVG-Animationen sind sehr praktisch, wenn der Browser-Support vorhanden ist

CSS-Präprozessore

Was sind CSS-Präprozessoren?

Wandeln CSS-ähnlichen Code in CSS um (Sass,PostCSS,Less,etc.). Sind nicht an CSS-Limitationen gebunden. Erlauben Modularisierung, Wiederverwendbarkeit und wartbaren Code.

Syntactically Awesome Style Sheets (SASS)

Erlaubt Sass- und SCSS-Syntax. Sass: "Einrück"-Syntax mit weniger Schreibaufwand. SCSS: Angepasster CSS-Syntax.

⇒ SCSS-Syntax wird bevorzugt. Sass-Syntax ist historisch bedingt.

Anwendung

Features: Variablen, Verschachtelung, Partials, Mixins, Vererbung, Operatoren, Funktionen, Maps. Weiteres: Mixins und Extends funktionieren ähnlich. Mixins generieren mehr Code-Redundanz, erlauben aber Input-Parameter. Extends sind sinnvoll bei thematischen Abhängigkeiten. Im Zweifelsfall Mixins verwenden.

⇒ Beide haben Vor-und Nachteile. Immer den CSS-Code im Auge behalten.
 ⇒ Hinweise: Max 3 Ebenen verschachteln. @use verhindert Überschreibungen

```
@use "colors";
@import "margins";
@mixin toPosition($x, $y: 15px) { // Mixins with Parameters
position: absolute; left: $x; bottom: $y;
@mixin onMobile() {
                                       // Mixins
  color: colors.$main-color;
                                       // With Namespace (use)
  li {
   margin: $main-margin;
&:hover {
                                       // No Namespace (import
// Parent-Selector (&)
       margin: 2 * $main-margin;
  }
}
a { color: blue; }
  div & f
                                       // Also Possible (&)
    @include toPosition(10px);
@include onMobile {
       .wide { display: none; }
 Wabstract-rule { margin: 0: }
                                       // Placeholder (not in CSS)
  @extend ul;
@extend %abstract-rule;
                                       // Inherits all ul-Rules
// Placeholder-Inheritance
                                                           stylesheet.scss
```

Smain-color: black; // Variables _colors.scss
Smain-margin: 19px; // Variables _margins.scss

```
OL, OL (color: black; )
ul 14, ol 14 (margin: 19px; )
ul 14; ol 14 (margin: 19px; )
ul 14:hover, ol 14:hover { margin: 28px; }
ul > a, ol > a { color: blue; }
div ul, div ol { position: absolute; left: 18px; bottom: 15px; }
div ul .vide, div ol .vide { display: none; }
div ul .narrow, div ol .narrow { display: block; }
ol { margin: 8; }
```

Logik und Rechnen

Einheiten: Numbers (1, 3.5, 5px), Strings ("foo", 'bar', px), Colors (blue, #04a3f3), Booleans, Nulls, Listen, Maps.

Rechnen funktioniert wie in der Physik: 2px+2px=4px, 2*1px=2px, 2px/2px=4px, 2*1px=2px, 2px/2px=2px, 2px*1px=Error (Kein px^2), 2px=2px, "2px"+1px=2px1px", "2px"*2=Error (String), 2px+1=3px.

eiteres

PostCSS: Ein CSS-Framework, welches Plugins für das CSS-Parsing und Preprocessing erlaubt (autoprefixer, minifer, etc.). Built-Tools: Automatisieren die Erstellung von App-Bundles (WebPack, Vite, Rollup, etc.). Optimieren Performance und Cross-Browser-Support. Beinhalten u.a. Tree-Shaking, Caching, Resource Loading Optimization. Polvfills und Transformationen (minity compress etc.)

⇒ Bekannte CSSFrameworks und Component Libraries: Bootstrap, Tailwind, Material UI.
⇒ Natives CSS-Nesting soll bald möglich sein (aktuell nur Chrome/Edge).

User-Centered

User-Researc

1st Rule of Usability: Höre nicht auf deine User, sondem schaue, was sie machen. Befragungen führen zu Spekulation & Wunschkonzert. Kunden dürfen befragt werden (Kunde ± Benutzer)

Wichtig: Benutzer, Aufgabe, Tool und Kontext beachten. Du bist nicht der Benutzer. User-Research ist immer möglich. Problem Space: Benutzer ist Experte. Kennt seine Aufgaben, Ziele und Probleme. Kann darauf analysiert werden. Solution Space: Designer ist Experte. Kann Lösungen für die Probleme entwickeln.

☼ Kunden und Benutzer müssen unterschiedlich analysiert und befriedigt werden.
Üser-Research ist aufwändig, aber unersetzlich für benutzerorientierte Produkte

Szenarios

Zeigen eine Geschichte, wie ein Problem aktuell (Problem-Scenario) und in Zukunft (Future-Scenario) gelöst wind. Dokumentieren die Verbesserungen durch das neue Tool. Werden durch Kunden & Benutzer validiert. Als Text oder Storyboard (Sketch, App.,etc.) möglich. Beinhalten: Benutzer (Persona), realistisches/beobachtetes Pro-

blem mit Kontext, Auslöser, Schritte, Lösung oder Fehlschlag.

Releases sollten sich an den Szenarien orientieren (anstatt den Features).

Befragungen

Grundsätzlich vermeiden. **Probleme**: Wer wird befragt? Wie wird gefragt (Offen, Suggestiv, etc.)? Was wurde zuvor gefragt?

Qualitative Studien: Wenige Befragungen, die viel aufdecken. Quantitative Studien: Viele Befragungen, die weniges validieren

Navigationsdesign

Mit guter Ausschilderung wissen Nutzer stets: Wo bin ich? Eindeutige Seitentitel, Headers, Icons, etc. und Venwendung von Breadcrumbs. Wo kann ich hin? Eindeutige und sichtbare Navigationselemente. Was ist passiert? Kontinuität und Aktionsfluss durch Animationen, etc. sicherstellen.

Qualität bestimmbar mit Kofferaum-Test. Welche Webseite? Welche Unterseite? Welche Hauptsektionen? Welche Navigationsoptionen? Wo im Gesamtkontext? Wo kann ich suchen?

⇒ mysite.ch/Vision/Our Goal, Inventor,.../Home, Team,.../About ... Vision/Oben Rechts.
⇒ Mt guter Ausschilderung fühlen sich die Benutzer gut aufgehoben (Ziel).

Methoden

Concept Model: Zeigt die Beziehungen zwischen Elementen (Klasse hat Lehrer). Site Map: Zeigt die Navigationshierarchie der Elemente (Klasse – Lehrer). Information Scent: Links sind so benannt, dass die erreichbaren Ziele erkennbar sind.

Card-Sorting: 1. Zielpunkte (Content Elements) definieren. 2. Gruppen mit 'Open Card Sort' bestimmen: 2.a 5+ Personen der Zielgruppe rekrutieren. 2.b Zielpunkte in disjunkte Gruppen unter teilen. 2.c Gruppen benennen lassen. 3. Gruppennamen bestimmen (Hypothese). 4. Gruppen mit 'Closed Card Sort' validieren: 4.a 5+ neue Personen rekrutieren. 4.b Gruppennamen vorgeben. 4.c Zielpunkte den Gruppen zuordnen lassen.

Tree-Testing: 1. Aktuelle Navigationsstruktur aufnehmen. 2. Szenarien definieren 3. Auffindbarkeit der Zielpunkte testen.

Usability-Test

Gute Szenarios mit sinnvollen Zielen definieren. Aufs wichtigste fokussieren (schnelle Iteration). Keine konkreten Elemente (Button-Texte, etc.) benennen. Richtige Personen rekrutieren. Vor- und Nach-Interview führen. Personen zum laut denken anregen.

Vorgehen: 1. Benutzergruppe bestimmen. 2. Testpersonen sammeln. 3. Aufgaben (Szenarios) durchspielen. 4. Probleme und Kontext analysieren. 5. Alles dokumentieren.

Isability-Standards

ISO 9241-11: Effektivität: Die Benutzer können ihre Ziele erreichen. Effizient: Der Aufwand zur Erreichung ist angemessen. Zufriedenheit: Die Benutzer sind gegenüber dem System positiv eingestellt.

180 9241-110: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit, Lemförderlichkeit

Hada

Gibt Sicherheit, erlaubt Korrektur und vermeidet unnötige Dialoge Hinweis auf Undo in passive Benachrichtigungen (Snackbar, Toast, etc.) einbauen. Undo immer mit Redo kombinieren.

Design-Erwägungen: Kontext (Alles, ein Feld, ein Upload, etc.).
Granularität (Buchstabe, Abschnitt, etc.). Operationen (Ausdrucken/Versenden nicht Undoable).

Weiteres

Wireframes/Mockups: Zeigen einen Sketch des Uls mit möglichst realistischem Inhalt (Papier, Figma, Azure, etc.). Können für Usability-Tests verwendet werden. Vereinfachung: Funktionen können entfemt, versteckt, gruppiert oder verschoben werden.