Bearifflichkeiten

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

Node is ist eine JavaScript Runtime mit einem riesigen Package Ecosystem. Vorteile: Server und Client in selber Sprache (Java-Script), nativer Support von JSON (z.B. für REST APIs), einfaches und schnelles Deployment, modularer Aufbau mit zahlreichen Pa ckages, wenig "Magie".

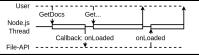
Node.js baut auf Chrome's V8 JavaScript Engine auf. Es ist aber kein Web-Framework und keine Programmiersprache

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

Node.is verwendet nur einen Thread. Wird ein Event ausgelöst

(z.B. 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 is 12 sind jedoch auch "Worker Threads" möglich.



Callbacks und Events

Callbacks sind Funktionen, welche zu einem späteren Zeitpunkt aufgerufen werden. Events sind Ereignisse, auf welche man sich registrieren kann

```
"s.readFile('demo.txt', (err, data) => { /* ... */ })
outton.addEventListener('click', (event) => { /* ... */ });
```

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

Promises

Sind eine Alternative zu Callbacks und lösen die Callback-Hell

```
fs.promises.readFile('demo.tx
.then(data => { /* ... */
.catch(err => { /* ... */
                                                                                       Variante 1
      const data = await fs.promises.readFile('demo.txt')
                                                                                       Variante 2
```

- Der aktuelle Standard in Node is sind immer noch Callbacks.

 Viele Module bieten aber eine Promise-Variante zusätzlich an (zB. fs. promises)

Module sind Code-Pakete, welche Funktionalitäten für andere Module bereitstellen. Node verwendet den Package Manager npm und kennt zwei Modulsysteme: CommonJS: Standard seit Beginn (module.js / module.cjs). ESM: Verwendbar seit ECMAScript 2015 und Node.js 14 (module.mjs).

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

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

⇒ Module werden vom System nur 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. package-lock.json: Beinhaltet die exakten Package-Abhängigkeiten. Wird automatisch aktualisiert und gehört ins Repo. Native Module: Beinhalten nativen Code. NVM: Versionsmanager für Node.js

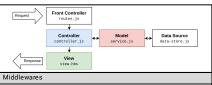
Was ist Express.is

Express.js ist ein bekanntes Web-Framework für Node.js. Es baut auf dem MVC-Pattern auf und verwendet Middlewares für die Request-Rearheitung

Model-View-Controller (MVC)

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

Ziel ist "Seperation of Concerns" (Alternativen dazu sind MVVM MVP.etc.)



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

⇒ Express.is stellt ab V4 viele Middlewares zur Verfügung (zuvor "Connect"-Plugin). Beispiele: BodyParser (bodyParser), Cookie-Parser (cookieParser), Cors, etc

Cookies/Sessions

Cookies: Repräsentieren ein kleines Stück von Daten. Der Server schreibt die Cookies zum Client, der Client sendet alle seine Cookies bei 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 Benut zer nun bei iedem Request mit dieser ID authentifizieren



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

Tokono

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 (z.B. 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, Ver wendung via fetch('/page').then(res => res.ison()).then(...)

Verwendung von Express.js

```
import express from 'express';
import bodyParser from "express";
import session from 'express-session
 import session from 'express-session'
import path from 'path';
import exphbs from 'express-handlebars'
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(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!');
```

```
import express from "express";
import { controller } from './co
const router = express.Router();
                                    ./controller.is'
 router.get('/books/:id/', controller.getBooks)
   .get(controller.getOrders)
.post(controller.postOrders) // Multiple callbacks possible router.all('/*', controller.default) // Uses pattern matching
export default router;
                                                                       routes is
    getOrders(req, res) { /* ... */ }
postOrders(req, res) { /* ... */ }
     postOrders(req, res) { /*
export const controller = new Controller():
                                                                    __controller.js
```

```
unction customLogSessionMiddleware(reg. res. next) {
   console.log(req.session.counter);
req.session.counter += 1;
   next(); /* Calls next middleware *.
export default customLogSessionMiddleware;
                                                    custommiddleware is
<html>
<head>
   <meta charset="utf-8">
<title>Example App</title>
 body>
{{{body}}}
</html>
```

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

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

```
import Datastore from 'nedb'
const db = new Datastore({ filename: './books.db',
                                                                     autoload: true });
db.insert({ name: 'A' }, (err, doc) => { /* - "/ };
db.find({ name: 'A' }, (err, docs) => { /* - "/ };
db.findOne({ name: 'A' }, (err, doc) => { /* - "/ };
db.update({ name: 'A' }, { name: 'B' }, { },
err, num] => { /* - "/ }; /
                                                                                                                                          data-store.js
```

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 (nnm. i. - D. @t.vnes/node)

Statt Node is kann auch ts-node verwendet werden (bietet JITCompilation von TS)

TS-Config

Strict Mode

nolmplicitAny: Keine untypisierten Variablen. alwaysStrict: Verwendet automatisch ein "use strict" im JS-File. strictNull-Chacke: null und undefined eind nicht mahr Teil der Basistynen (explizite Deklaration nötig), strictFunctionTypes: Strenge Überpriifung von Funktionstynen strictPropertylnitialization: Klassen eigenschaften müssen initialisiert werden

⇒ Weitere Finstellungen sind nolmplicitThis und strictBindCallApply

Spezifikation

Basistynen

boolean, number, string, null: Wie in anderen Sprachen. undefined: Variable deklariert, aber kein Wert zugewiesen, any: Beliebiger Wert Zuweisung in heide Richtungen beliebig möglich, unknown: Unbekannter Typ. Zuweisung zu unknown beliebig möglich, Zuwei sund von unknown erst nach Type Narrowing Type Inference: Ohne Tyndeklaration wird der Tyn automatisch hestimmt

```
let aNumber: number = 1
let aString = 'abc' // Inferred string
 let aUndefined: undefined;
let aAny: any = true
 aNumber = aString
 aNumber = aAny
aAny = aString
 aUndefined = undefined // OK
  aString = undefined
                                 // NOK (in Strict Mode)
```

Komplexe Typen und Typdeklaration

Type Narrowing

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

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

```
enum EnumType { A, B, C }
type UnionType = number | undefined
 type StrLitUnionType = 'A' | 'B' | 'C'
let aUnionType: UnionType;
let anArray: number[] = [1, 2, 3]
let anUnionArray = [1, 'abc'] // Inferred (number|string)[]
let aTupel: [number, string] = [1, 'abc']
let anEnum = EnumType.A
let aStrLit: StrLitUnionType;
 aUnionType = undefined
aUnionType = 'abc' /auUnionArray[0] = 'abc' /auUnionArray[0] = 'abc' /astrLit = 'B' /a
 aStrLit = 'abc'
```

TyneScript verwendet Elussanalyse um die tatsächlichen Tynen 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
aNumber = aUnknown
aNumberOrString = 1
aUnknown = 1 // OK
aNumber = aNumberOrString // OK
aNumber = aUnknown // NOK
if (typeof aUnknown === 'number'){
       aNumber = allnknown
```

Funktionen

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)
```

Klassen und Interfaces

Wie in anderen Sprachen. Properties lassen sich im Konstruktor definieren (automatische Generierung und Initialisierung). Generics sind möglich. Mit Readon I vers können alle direkten Felder einer Klasse T unveränderlich 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: numb
  private val2: string
   static readonly VAL4 = 1 // Public, Static, Readonly
   constructor(val1: number, val2: string, val3?: number) {
    this.#val1 = val1; this.val2 = val2; this.val3 = val3;
  set vali(val: number) { this.#vali = val } // ES6
get vali() { return this.#vali } // ES6
 interface AInterface<T> {
    readonly valA: T // Public (Never static, private,
      valB?: boolean
class ASubClass<T> extends AClass implements AInterface<T> {
      constructor(public valA: T, private valC: number) {
    super(1, 'abc')
       func() { /* ... */ }
let aClass = new AClass(1, 'abc', 2):
let aSubClass = new ASubClass<number>(1, 2);
aSubClass.val1 = 1;
let aIntA: AInterface<number> = { valA: 1, func() {} } // OK
let aIntB: AInterface<number> = { valZ: 'abc' } // NOK
// Type Assertions
let aTypeA= aSubClass as AClass; // OK
```

Weiters

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' | '
type TempLit = 'my-${Keys}` // type TempLit = 'my-x' | 'my-
```

esponsive Layout

Flexible Layout: Verfügbarer Platz wird durch Verbreiterung der Elemente ausgefüllt. Responsive Layout: Verfügbarer Platz wird durch Umordnung oder mit neuen Elementen ausgefüllt (benötigt Media-Oueries), Graceful Degradation: Fokus auf neue Browsers. Alle Features nutzen und dann Rückwärtskompatibel machen (Polyfills, etc.) Progressive Enhancement: Fokus auf alte Browser Mit Grundfunktionen starten und dann neue Features zusätzlich anbieten. Mobile-First: Fokus auf Mobile statt Desktop (z.B. Base-CSS für Mobile)

ImNormalfall werden Flexible Layouts und Responsive Layouts miteinander kombinier Wichtig, da es zahlreiche Geräte mit unterschiedlichen Screen-Sizes gibt

Normalize & Reset

Problem: Alle Browser haben ein etwas anderes Default-Styling. Reset: Entfernt die meisten Default-Styles. Entwickler muss Kon sistenz schaffen Normalize: Entfemt ausschliesslich inkonsis tente Default-Styles. Konsistenz so gewährleistet

Media-Queries

Erlaubt CSS für unterschiedliche Medien/Dimensionen.

Typen: screen, print, Dimensionen; width, min-width, max-width, height.min-height.max-height. Zustände: orientation: landscape hover: hover|none, pointer: fine|coarse|none. Operatoren: and, not, only, , (or). Achtung: and bindet stärker als ,. Featureabfrage: @supports (display: grid) {...}.

→ Typische Breakpoints: Bis 480px/30em; Smartphones, Bis 768/48em: Tablets. Bis 992px/62em: Laptops. Danach Desktops. ⇒ CSS verwendet CSS-Pixel (px). Das Verhältnis zu echten Pixel ist abhängig vom Zoom/Gerät.

```
body { color: blue; }
| 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; }
```

Media-Oueries können im link media=...>-Tag gesetzt werden

Mit <meta name="viewport" content="width=device-width,initialscale=1"> wird das "intelligente Zoomen" von Mobile-Browsern un terbunden. Wichtig, damit die Media-Queries greifen

CSS-Features

 $\textbf{Box-Modell (box-sizing): } \underline{\textbf{content-box}} \ \textbf{bedeutet, Boxgrösse ist width}$ + 2*padding + 2*margin und Inhalt ist width. border-box bedeutet, Boxgrösse ist width und Inhalt ist width - 2*padding - 2*margin

Flement Left/Right Margin/Padding

Top/Bottom Margin/Padding Width/Height Causes Line-Break Allows Scroll

Adds White-Space inline

block inline-block

Prozent (%): Bezieht sich immer auf das Parent-Element, ausser bei translate(x, y). Viewgrösse (vh/vw): Prozentual zur Viewgrösse. Achtung: 100vh ist manchmal grösser als der Screen (z.B. Browserheader). Besser 100dyb verwenden. Funktionen: calc(100yb - 5px), min(..., ...), max(..., ...), clamp(min, ideal, max), Scrollen (overflow) hiddenty is the Iscrelliante Scrollhar soll erkennhar sein (nicht immer der Fall). Position (nosition): absolute ist relativ zum ersten Parent mit position: absolute trelative, fixed ist relativ zum. Browserfenster, sticky ist wie absolute, bleibt aber oben/unten am Bildschirmrand haften. relative ist relativ zur aktuellen ("realen") Position. static ist normal im Fluss.

Custom Properties

CSS erlaubt Definition von Variablen. Werden bei der ersten Verwendung im DOM (Root-to-Leaf) ausgerechnet (Aufpassen bei "berechneten" Werten). Können in JS mit style, getprogertyvalue(.) und style.setProperty(__) ausgelesen/verändert werden.

```
:root { --main-color: yellow
                    -calculated: var(--main-color); }
              --catculated: var(--main-color);
color: var(--main-color);
background: var(--calculated) }
--main-color: blue;
color: var(--main-color);
background: var(--calculated) }
h1 {
p {
```

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:row|column|row-reverse|column-reverse. Umbruchverhalten: flex-wrap:nowrap|wrap. Hauptachsen-Ausrichtung (pro Zeile): justify-content:flex-start|flexend|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.

⇒ flex-wrap beachtet flex-basis, width, height und ignoriert flex-shrink ⇒ flex-wrap: wrap garantiert keine exakte "Grid"-Ausrichtung der Elemente.

Achtung: row bedeutet z.B. 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". flex-grow: 2 ist 2× so gross wie flex-grow: 1.0 bedeutet kein Wachstum

Container/Parent

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

Anzahl und Grösse von Zeilen/Spalten: grid-template-co lumns|grid-template-rows: [measure]. Row-Ausrichtung (pro Zeile): justify-items: startlendicenteristretch, Column-Ausrichtung (pro Zeile): align-items: [justify-items]|baseline.

Grid-template-rows: 10px 80% 1fr (3 Zeilen mit definierter Höhe) Berechnung: Grid-Grösse: 100px mit 10px 1fr 2fr ⇒ 90px freier Platz $\Rightarrow 90/(1+2) = 30px \Rightarrow 1fr = 30px, 2fr = 60px;$

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

Item/Child

Positionsbestimmung: grid-column-start: x₁, grid-column-end: x₂, grid-row-start: y₁, grid-row-end: y₂, grid-column: y₁/y₂, grid-row: y₁/y₂, grid-area: y₁/x₁/y₂/x₂. Row-Ausrichtung: justify-self: start|end|center|stretch. Column-Ausrichtung: align-self: [jus tify-self]. Row+Column-Ausrichtung: place-self: [justify-self].

Werte

[number]fr: Fraktion des verfügbaren Platzes. Erlaubt Dezimal. Können 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. fit-content: [length]: Entspricht minmax(auto, [length]). repeat([number]|auto-fill|auto-fit, [measure]): Wiederholt Wert so oft wie angegeben. auto: Automatische Grössenanpassung.

Rechtliches

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

Wichtig für Menschen mit Seh- Hör- kognitiven oder motorischen Behinderu Themen

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 Rild Kontext Inhalt und allenfalls Text als Zitat an geben. (A comic strip 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 (z.B. bei ADHS), Epilepsie und Migräne zu verhindern (@media (prefers-reduced-motion) { }). 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 flev-direction: reverse etc.)

Screenreader: Für Unterstützung keine Heading-Levels auslassen, semantische Elemente verwenden (anstatt ARIA-Attribute), versteckte Skip-Links ("Zum Hauptinhalt") am Seitenanfang ein hauen Lang-Attribut setzen Tabellen-Headings für Zeilen/Snalten 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 AccessibilityTests sind limitiert. Manuelle Tests immer emnfohlen Top-5 Probleme: Kontrast, kein Alt-Text leere I inke keine I shele leere Distron

Wer braucht was?

Sehbehindert: Kontraste, Zoombarkeit, Trennung von Inhalt und Layout. Hörbehindert: Audiotranskription, visuelles Feedback, einfache Texte. Blind: Gute Dokumentstruktur. Alt-Texte. Tastatur bedienung. Motorische Einschränkung: Grosse Schaltflächen, Tastaturbedienung, keine ungewollten, automatischen Aktionen Lem- & Aufmerksamkeitsstörung: Einfache Texte, lesbare Schriftarten, keine ablenkenden Flemente

⇒ Retrifft 21% der Schweizer Revölkerung

OWASP Top 10

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

Fokusthemen

A01.1: Insecure Direct Object References (IDOR)

Reschreibung: Eine Webseite ist verwundbar wenn Daten ohne Sicherheitsmechanismen über direkte Referenzen erreichbar sind. Beispiel: Die persönlichen Daten eines Nutzers lassen sich über my-site.ch/:usertp einsehen, ohne dass sichergestellt wird dass der Aufrufer auch die entsprechenden Rechte hat. Ein Angreifer kann nun z.B. verschiedenen 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 reg.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 night übergrüft. Reisniel: Ein Angreifer bringt einen eingeloggten User dazu, auf evil.ch zu gehen. Auf dieser Seite wird (automatisch) ein Formular an my-site.ch/deleteAccount gesendet Da der Browser die Cookies der realen Webseiten mitsendet wird die Aktion im Namen des Nutzers ausgeführt.

Lösung: Formulare mit einem CSRF-Token versehen und beim Request überprüfen. Bei Express.js z.B. das csurf Plugin verwenden. Alternativ keine Cookies verwenden (sondern z.B. JWT).

A01.3: Replay Attacks

Reschreibung: Fine Webseite ist verwundhar wenn Aktionen unbeabsichtigt mehrmals ausgeführt werden können.

Beispiel: Eine Webseite belohnt einen Nutzer für jede korrekte Aufgabe mit Punkten. Der Nutzer kann nun eine einzige Aufgabe mehrmals absenden und erhaltet so mehrere Punkte.

Lösung: Keine "generische" Lösung möglich. In diesem Fall z.B. die Aufgaben abspeichern 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 Reisniel: Eine Wehseite sendet die Logindaten eines Benutzers unverschlüsselt und ohne HTTPs an den Server. Ein Angreifer kann nun 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 in ein Formular werden ohne "Escaping" an andere Nutzer ausgeliefert. Ein Angreifer kann nun z.B. eingeben und so Code auszuführen

Lösung: "Escaping"/"Encoding" verwenden ({{content}}} statt {{{content}}}} in HBS), Fingabebereinigung ("Sanitizing") via Libra ries (XSS, DOMPurify) einbauen, CSP-Header setzen, HTTPOnly-

A03.2 Remote Code Execution / Injection

Beschreibung: Ein Server ist verwundbar, wenn ein Angreifer den Server dazu bringen kann, seinen Schadcode (z.B. JavaScript. SOL. 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()) ausführen.

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

setTimeout(_) und setInterval(_) verhalten sich ähnlich wie eval(_)

A07: Identification & Authentication Failure

Reschreibung: Fine Webseite ist verwundhar wenn sie u.a. Brute-Force-Attacken erlaubt, schlechte Passwörter zulässt oder keine Multi-Faktor-Authentisierung verwendet.

Beispiel: Ein Angreifer bekommt Zugriff auf ein Nutzerkonto, indem er alle möglichen Passwortkombinationen durchprobiert. Da das Passwort 1234 war, ging die Attacke nur wenige Minuten.

Lösung: Authentisierung korrekt umsetzen oder externen 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 CySec-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). HTTPOnly- Secureund SameSite-Cookies: Blockiert den Zugriff von Client-Scripts

(app.use(csurf({cookie: true}) und im Formular <input name= csrf value={{csrfToken}}). Keine Cookies verwenden (stattdessen z.B. JWTs).

Testarten

Static: Statische Code-Analyse. Unit: Testen einer einzelnen Komponente (Klasse, Modul, etc.). Integration: Testen von mehre ren Komponenten. E2E/System: Testen von allen Komponenten. Funktional: Testen von konkreten Anforderungen (Use-Cases). Regression: Testen auf notenzielle Fehler nach Code-Änderungen. Weiteres: Testen von Security, Usability, Performance, Stress, ⇒ Oft wird dabei von einem SUT (System Under Test) gesprochen.

Phasen: 1. Setup, 2. Exercise, 3. Verify, 4. Teardown

das Testen von Web-UIs (Cypress, Puppeteer).

mittels Dependency Injection zulassen.

⇒ Achtung: zT. wird auch das Test-Environment als «Fixture» bezeichnet

Assertion Libraries: Frlauben eine einfache Spezifikation von

(Mocha, Cypress, Jest), Mocking Libraries; Erlauben die Erstel-

lung von Mocks (Proxyguire, Sinon, is), DOM-Handling; Erlauben

Alles Testen, das kaputt ging oder gehen könnte. Neuer Code ist

immer (schuldig), bis das Gegenteil bewiesen wurde. Vor einem Push ins Repo immer alle Tests laufen lassen. Mocks/Doubles

Hard-to-Test Code, Production Bugs, Fragile Tests (Tests sind zu

stark von interner Logik abhängig), Erratic Tests (manchmal er-

on Roulette (zu viele Assertions in einem Tests), Test Logic in

Beginnt mit expect(...) .to.egual(1) ..to.be.false.to.not.be.undefined, to.deep.equal(array), to.be.a(number), to.throw(TypeError), to.be.an(array).that.does.not.include(3), to.have.any.key(key)

it('Unit: Changed value is new value', () => {
 aNumber = 2
 // Exercise

})
it('Integration: Title contains id of book', () => {
 chai.request(app).get('/books/42').end((err, res) => {
 const dom = new jsdom.JSDOM(res.text)

Internationalisierung (I18N): Programmieren auf eine Art, sodass

(G11N): Anpassung des Programms (Sprache, Farbe, etc.) an die Sprachregion. Übersetzung (T9N): Übersetzung von Texten und

Wörtern, Locale: Bezeichnung der Sprachregion (z.B. als String).

Layout: Elementpositionen an kulturelle Benutzergruppe anpas-sen (z.B. Leserichtung), Textlänge soll Hierarchie nicht beeinflus:

sen (ungünstige Umbrüche). Lavout nicht von der Wortsortierung

abhängig machen. **Eingabefelder**. Unterschiede bei Postleitzahl

Name anstatt Vor- und Nachname verwenden. Unterschiedliche

Anführungszeichen: Werden bei <q>_</q> automatisch basierend auf lang-Attribut gesetzt. Standard: ES2021 Internationalisierung.

const regions = new Intl.DisplayNames(['en'], { type: 'region' })

dFormatter.format(Date.now()) // 7/9/2023
new Date(Date.now()).toLocaleString(['en-US']) // Alternative

const lFormatter = new Intl.ListFormat(['en'])
lFormatter.format(['Eggs','Milk','Fish']) // Eggs, Milk, and Fis

const swiss = new Intl.Locale('gsw') // Alternative zum String

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

Datums- und Zahlenformate zulassen.

Bibliotheken: FormatJS, Polyglot.js, i18next.

const collator = new Intl.Collator(['de'])
collator.compare('Äh?', 'Zzz') // Smaller (-1)

tFormatter.format(-1, 'week') // 1 week ag

Formatter.format(1250.50) // 1.256

const nFormatter = new Intl.NumberFormat(['de'])

const plural = new Intl.PluralRules(['en-US'])

olural.select(1) // 'one': e.g. write '1 cat' olural.select(10) // 'other': e.g. write '10 cats

const dFormatter = new Intl.DateTimeFormat(['en-US'])

const tFormatter = new Intl.RelativeTimeFormat(['en'])

Telefonnummer, Provinzbezeichnung, etc. beachten. Vollständige

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

Lokalisierung möglich wird. Lokalisierung (L10N) / Globalisierung

expect(dom.window.document.guerySelector('h1')).to.com

expect(aNumber).to.equal(2) // Verify

folgreich, manchmal nicht), Developers not Writing Tests, Asserti-

Production Code, Obscure Tests (zu kompliziert), Slow Tests, Test Code Duplication. Conditional Test Logic (Codeteile werden

Tests (Chai, Expect.is), Test-Runners: Führen die Tests aus

⇒ Assertion Library könnte mit throw new Error (_) ersetzt werden (Sinnlos)

Struktur von Unit-Tests

Tools

Prinzipen

nicht ausgeführt)

Verwendung (Chai)

import chai, {expect} from 'chai'
import chaiHttp from 'chai-http';

describe('My Test-Fixture', () => {

import jsdom from 'jsdom'; import app from './app.js'

let aNumber; beforeEach(() =>{

afterEach(() => {

})

Regriffe

Best Practices

aNumber

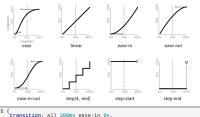
Test-Gruppe/Fixture: Beinhaltet die Tests, Test-Environment: Umgebung, in welcher die Tests ausgeführt werden. Doubles/ Mocks: Fake-Klassen fürs Testen

Transitions

Erzeugen einen weichen Übergang zwischen zwei Zuständen. Die Frames dazwischen werden automatisch ausgerechnet

transition-property: CSS-Property zum Animieren. transition-duration: Animationsdauer. transition-timing-function: Beschleunigungsfunktion. transition-delay: Zeitdauer, bevor die Animation

Kurzschreibweise mit transition. Erlaubt auch mehrere Übergän-



color is linear 0.5s;

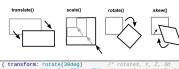
⇒ Alle Animationen werden über eine Cubic Bezier Curve definiert.

background: red; /* See 'all' */

Transform

color: white:

Verändert die Form und Position eines Elements, wenn es angezeigt wird. Kann über Transitions animiert werden. Rotationen sind im Uhrzeigersinn. Der Ursprung kann mittels transform-ori gin: horizontal vertical; festgelegt werden. Bei 3D-Animationen lässt sich auf dem Parent-Element das perspective-Property setzen (ie tiefer der Wert, desto extremer die Perspektive).



```
rotate(30deg)
translate(20px, 5%) /* translateX, Y, Z, 3d */
coale(1 5) /* scaleX, Y, Z, 3d */
      skew(15deg);
transform-origin: left top;
transition: transform 300ms ease; }
/* Looses translate and scale! */
p:hover { transform: rotate(60deg) skew(0deg); }
```

 Bei mehreren Transforms ist die Anordnung relevant Achtung Rei Zustandswechsel immer alle gesetzten Transforms wiederholen

Keyframe Animationen

Erlaubt die Animation von einer Serie von Zuständen

```
@kevframes mv-animation {
       0% { background: red; }
50% { background: green; }
100% { background: blue; } }
h1 { animation-name: my-animation;
         animation-timing-function: linear;
animation-iteration-count: 3; /* Or infinite */
animation-direction: normal; /* Or revert, alternate */ }
```

Die meisten Properties mit quantitativen Werten oder Farben lassen sich animieren. Nicht quantifizierbar sind z.B. border-style, display, auto, url(...), etc.

Mit Houdini (@property) lassen sich neu auch solche Properties animieren

Weiteres

JS-Animationen: Mächtiger, aber nicht performant (CSS bevorzugen). SVG-Animationen: Sehr praktisch, wenn Browser-Support vorhanden

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

Syntactically Awesome Style Sheets (SASS)

CSS-Präprozessor. 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, erlauhen aber Innut-Parameter, Extends sind sinnvoll hei thematischen Abhängigkeiten. Im Zweifelsfall Mixins verwenden.

```
Beide haben Vor- und Nachteile. Immer den CSS-Code im Auge behalten.
  Maximal bis zu 3 Ebenen verschachteln (Lesbarkeit) und "use" bevorzugen (Über
@mixin toPosition($x, $y: 10px) { // Mixins with Parameters
   position: absolute;
   left: $x; bottom: $y;
@mixin onMobile() {
                                          // Mixins
// Content of Importer
 @content;
& .narrow { display: block; }
                                         // Reference to Importer (8)
    olor: colors.$main-color;
                                          // With Namespace (use)
                                          // Nesting
// No Namespace (import)
    margin: $main-margin;
   &:hover {
  margin: 2 * $main-margin;
   a { color: blue: }
 div & {
  @include toPosition(10px);
                                        // Also Possible (&)
// Mixins
    @include onMobile {
   .wide { display: none; }
%abstract-rule { margin: 0: }
                                        // Placeholder (not in CSS)
 @extend ul;
@extend %abstract-rule;
                                                               stylesheet.scss
Smain-color: black:
                                                                   _colors.scss
$main-margin: 10nx:
                                                                 _margins.scss
```

Logik und Rechnen

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

result.css

Rechnen funktioniert wie in der Physik!

ul, ol { color: black; }
ul tl, ol ti { margin: 19px; }
ul ti:hover, ol ti:hover { margin: 28px; }
ul > a, ol > a { color: blue; }
div ul, div ol { position: absolute; }
left: 19px;
hertrail 18px; }

bottom: 10px; }
div ul .wide, div ol .wide { display: none: }

div ul .narrow, div ol .narrow { display: block; }
ol { margin: 0; }

```
$map: ("red": red, "blue": blue); // Maps
@function do($val) {
 @if $val == 2px {
    @return 2 * $val;
} @else {
                                    // 2 * 2px = 4px
    @return $val;
Meach Selem in Slist /
 // Creates text-red {...} and text-blue {...}
@mixin color-modifiers {
 @each $key, $val in $map {  // Loops with Maps
&-#{$key} { color: $val; }  // Template String
.text { @include color-modifiers; }
                                                       calculations.scss
```

PostCSS: Framework für CSS. Erlaubt Plugins fürs CSS-Parsing/ Preprocessing (z.B. autoprefixer minifier 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, Polyfills und Transformationen (minify, compress, etc.).

Bekannte CSS-Frameworks: Bootstrap, Tailwind, Material Ul. Natives CSS-Nesting soll bald möglich sein (aktuell nur Chrome/Edge

User-Centered Design

1st Rule of Usability: Höre nicht auf deine User, sondern 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, Probleme. Kann darauf analysiert und interviewt werden. Solution Space: Designer ist Experte. Kann Lösungen für die Probleme entwickeln.

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

Szenarios

Zeigen eine Geschichte, wie ein Problem aktuell (Problem-Scenario) und in Zukunft (Future-Scenario) gelöst wird. Dokumentieren die Verbesserungen durch das neue Tool. Werden durch die Kun den und Benutzer validiert. Als Text oder Storyboard (Sketch, App. etc.) möglich.

Beinhalten: Benutzer (Persona), realistisches/beobachtetes Problem mit Kontext, Auslöser, Schritte, Lösung oder Fehlschlag.

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

Befragungen

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

Qualitativ: Wenige Befragungen, die viel aufdecken. Quantitativ: Viele Befragungen, die weniges validieren.

Navigationsdesign

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

Qualität bestimmbar mit Kofferraum-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.
⇒ Resultat: Benutzer fühlen sich gut aufgehoben.

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 unterteilen. 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: Benutzergruppe bestimmen, Testpersonen sammeln, Aufgaben (Szenarios) durchspielen, Probleme und Kontext analysieren. (Alles Dokumentieren)

Usability-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 einnestellt

ISO 9241-110: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit, Lernförderlichkeit

Undo

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 der Benutzeroberfläche mit möglichst realistischem Inhalt (Papier, Figma, Axure, etc.). Kann für Usability-Tests verwendet werden. Vereinfachung: Funktionen können entfernt, versteckt, gruppiert oder verschoben

⇒ Weitere Tools (Gimbal Flowmann)