Web Engineering 2

1 Node.js

Callback: Funktion, die als Argument einer anderen Funktion übergeben wird, wo sie aufgerufen wird

function logger(msg) { console.log(msg); } function fn(n1, n2, callback) { let sum = n1 + n2; callback(sum); } fn(1, 2, logger);

Promise: bildet asynchrone Abläufe besser ab, bei Erfolg wird Fulfilled-Cb. aufgerufen, bei Fehler Reject-Cb.

let prom = new Promise((resolve, reject) => { setTimeout(() => { let success = true; if (success) { resolve("good"); } else { reject("bad"); } }, 2000); }); prom.then(msg => console.log(msg)) .catch(error => console.log(error));

Wichtige Node Packages (mehr Methoden vorhanden)

const fs = require('node:fs/promises'); // oder node:fs readFile(path[, options]); writeFile(file, data[, options]); unlink(path); mkdir(path[, options]); rmdir(path[, options]) // vor Funktion jeweils "fs.", callback letztes Argument const url = require('node:url'); const u = new URL("1"); // z.B. u.href & u.search.guery

https://juser[:|pass|@|sub.host.com|:|8080|/p/a/t/h?|query=string|#has protocol username password hostname port pathname search has

const http = require('node:http'); http.createServer(function (req, res) { res.writeHead(200, {'Content-Type': 'text/html'}): res.setHeader('name', 'value'); res.statusCode = 200; res.statusMessage = 'Ok'; res.write(req.url); res.end(); }).listen(8080); const path = require('node:path'); const filename=path.basename('/a/b/f.js'), abspath= path.resolve('dir', 'f.js'), ext=path.extname('f.js'); // .js

CommonJS: älter, verbreiteter, synchron, nur in Node. dynamische Imports at Runtime, kein Tree Shaking, File Extensions .is/.ts/.cis/.cts. use strict muss explizit gesetz werden, Import mit require, this=exports

module.exports = // default exports = // default read-only module exports name = // named exports.name = // named read-only

ESM: ab Node 14, asynchron, in Node & Browser, statische Imports at parse time, File-Extensions .mjs/.mts, strict Mode ist Default, this=undefined, Resolve-Reihenfolge: 1. Core Module (z.B. from 'fs'), 2. Pfad mit /, ./, ../ am Anfang (z.B. from './m.mjs'), 3. Filename (z.B. from 'counter', Modul wird in node_modules bis root gesucht)

import defaultStuff, { namedStuff } from 'file path'; export default name // default export name // named

package.json: beinhaltet Projekt-Infos, nötig für Publishing & Installation, definiert Scripts, package-lock.json beschreibt exakten Abhängigkeitsgraph

{ "name": "test", "version": "1.0.0", "main": "index.js". "scripts": { "test": "echo \"no tests\" && exit 1" }, "keywords": [], "author": "", "license": "ISC", "description": "", "type": "commonjs/module"]

2 Express.js

meistgenutztes, etwas veraltetes Web Framework

import express from 'express'; const app = express(), router = express.Router(); app.use('/static', express.static(dirname + '/public')); app.use(router); app.listen(3000, '127.0.0.1', function() console log('App listening on port 3000!'); });

```
Model
HTTP Respon
```

Middlewares: werden mit use() registriert, Registrierungsreihenfolge = Ausführungsreihenfolge, next(); zeigt eweils auf die nächste MW, offizielle Middlewares: body parser, compression, connect-timeout, cookie-parser, cookie-session, csurf, errorhandler, express-session, me thod-override, morgan, response-time, serve-favicon, serve-index, serve-static, vhost

Error-Middleware: bearbeitet Errors, welche von den MWs generiert wurden, sollte zuletzt registriert werden, mehrere sind möglich, wenn next() mit einem Error-Objekt aufgerufen wird, werden keine MWs mehr ausgeführt und stattdessen die Error-MWs ausgeführt, 4 Pa-

app.use(function(err, reg, res, next) { console.error(err.stack); res.status(500).send('error'); })

Body-parser: parst Request-Bodies und stellt die Variablen unter reg.body zur Verfügung

const bodyParser = require('body-parser'); app.use(bodyParser.json()); app.post('/saveData', (req, res) => { console.log("Using Body-parser: ", req.body.email)});

Cookie-parser: Parst den Cookie Header und stellt die Cookies unter req.cookies zur Verfügung

const cookieParser = require('cookie-parser'); app.use(cookieParser()); // opt. secret & options Arg app.get('/', function (req, res) { console.log(JSON.stringify(reg.cookies)); console.log(JSON.stringify(req.signedCookies)); res.cookie("url", reg.url);

res.cookie("signedUrl", req.url, {signed: true}); }); Express-session: reg.session ist ein Objekt, das tempoär Infos über aktuelle User Sessions speichert

const session = require('express-session'); const s={secret:"x",resave:true,saveUninitialized:true}; app.use(session(s)); app.post("/incr", (req, res) => { const session = req.session; session.count = (session.count || 0) + 1;

Error-handler: development-only, schickt error stack traces zurück zum Client je nach Konfiguration

res.status(200).end("" + session.count); });

const errorhandler = require('errorhandler'); f (process.env.NODE ENV === 'development') { app.use(errorhandler({ log: errFn }));}//leeres Arg mögl function errFn(err, str, req, res) { ... }

Csurf: verhindert Cross-Site Request Forgery (CSRF) Angriffe durch CSRF Tokens in Cookies

const csrf = require('csurf'); let csrfProtection = csrf({ cookie: true });

let prsFrm=bodyParser.urlencoded({extended: false}); app.use(cookieParser());

app.get('/form', csrfProtection, function (req, res) { res.render('login', { csrfToken: reg.csrfToken() }); }); app.post('/process', prsFrm, csrfProtection, function (req, res) { res.send('success'); });

express.static: Bereitstellung statischer Files wie Bilder CSS-/JS-Files, Beispiel am Anfang des Kapitels express.router: Router fürs Request-Handling, Beispiel router const und use(router) am Anfang des Kapitels

router.all('/', function (reg, res, next) { console.log("Router Working"); res.end(); }) router.post('/user', function (req, res, next) { ... }); Request Parameter: Ressourcenidentifikation, Abbildung hierarchischer Datenstrukturen

app.get('/users/:uid/items/:iid', (reg, res) => { const uid = req.params.uid, iid = req.params.iid; ... });

URL Ouerv Parameter: Such-/Filter-Operationen. Steuerung Pagination, Beispiel für "/search?t1=a&t2=b" app.get('/search', (req, res) => {

const t1 = req.query.t1, t2 = req.query.t2; ... }); NEDB: dokumentorientierte, file-backed Datenbank import Datastore from 'nedb-promises' export class TodoStore { db: Datastore; async add(...) {const r = this.db.insert(...); return r; } async update(id, ...) { await this.db.update({_id: id},

{\$set: {...}}; return this.get(id); } async delete(id) { await this.db.update({ id: id}, {\$set: {"state": "DELETED"}}; return this.get(id); } async all() { return this.db.find({}); } }

Cookies: im Browser, können persistent sein, automatisch bei jeder Anfrage gesendet, für Sitzungsverwaltung Personalisierung, Tracking und Analyse Session Objekte: auf dem Server, Lebensdauer bis Ab-

lauf oder Logout, nur innerhalb von Express verfügbar. für Sitzungsverwaltung und speichern temporärer Infos iber aktuelle User Sessions

REST: Verschiedene Paths für verschiedene Ressourcen. HTTP-Verben & JSON korrekt verwenden, stateess: Session durch JWT Token im Header ersetzen

res.format({'application/json':function(){res.send(true);}} const hdrs = new Headers(); hdrs.append('Content-Type/Accept', 'application/ison'); const init = { method: 'GET', headers: hdrs, cache: default', credentials: 'include', body: JSON.stringify(d)); const req = new Request('https://api.ch/users', init); fetch(reg).then(response => return response.json())

Neben json(): arrayBuffer(), blob(), formData() & text() Web-Sockets: bidirektionale, dauerhafte Kommunikationsverbindung zwischen Client & Server, geringe Latenz wenig Overhead, komplexe API (socket.io), für Echtzeit-Apps: Chat. Game, Live-Ticker, Docs-Collab, Push-Nachrichten, interaktive Video- / Audio-Streams Handlebars.is: Template Engine, um auf der Server-

Side HTML Code zu generieren

.then(data => ...);

import exphbs from 'express-handlebars'; mport path from 'path':

import { helpers } from './utils/handlebar-util.is' import { routes } from './routes/routes.is': const hbs = exphbs.create({ extname: '.hbs',

defaultLayout: "default", helpers: { ...helpers } }); app.engine("hbs", hbs.engine);

app.set("view engine", "hbs"); app.set("views", path.resolve("views")); app.use(routes); / controller/controller.js

export class Item{constructor(n){this.name=n}} export class Controller { showIndex = (req, res) => { res.render("index", new Item("itemname")); }; } export const controller = new Controller();

/ routes/routes.js

import express from "express"; import { controller } from "../controller/controller.is": const router = express.Router();

router.get("/", controller.renderIndex); export const routes = router;

// utils/handlebar-util.js export const helpers = { if eq': function (a, b, opts) {

if (a === b) return opts.fn(this): else return opts.inverse(this); } } {{! views/lavouts/default.hbs }} <!doctype html><html lang="en"> <head>...</head><body>{{{body}}}}</body></html> {{! views/index.hbs }}

{#if eq name "itemname"}}{{name}}{{/if eq}} Normaler Helper: {{upper name}} ${\text{each list}}{{this.name}}{{@root}}{{/each}}$

Weitere Hbs Builtin Block Helpers: #if, #unless, #with

3 TypeScript

TypeScript-ESLint: statische Analyse des Codes zur ehlerfindung, Installation: "npm install --save-dev eslint-@eslint/js @types/eslint__js typescript typescript-eslint", Lint ausführen: "npx eslint .", eslint.config.js:

/ @ts-check

mport eslint from '@eslint/js'; import tseslint from 'typescript-eslint'; export default tseslint.config(

eslint.configs.recommended,

...tseslint.configs.recommended, // wenn kein strict/styl ...tseslint.configs.strict, ...tseslint.configs.stylistic);

Prettier: formatiert Code beim Speichern, setzt konsistente Stilregeln durch, ESLint berücksichtigt auch Prettier Regeln, VS Code Extension, .prettierrc:

["singleQuote": true,"trailingComma": <mark>"all</mark>","tabWidth": 2]

Basis-Typen: boolean, number, string, null, undefined (no value), any (don't care), unknown (don't know), void, never, array, tuple, enum, union declare let v3: string; // globale Var aus nicht-TS-File

let v1 = "Hayes"; // Type by inference let v2 : number; v2 = 1; // Statically typed interface User { name: string; readonly id: number; } class UserAccount implements User { constructor(public name: string, public id: number) {} } const user: User = { name: v1, id: v2 };//structural typin const user = { name: v1, id: v2 }; // duck-typing function deleteUser(user: User) { ... } function getAdminUser(): User { ... } type WindowStates = "open" | "closed" | "minimized";

type WindowStatesLong = `\${WindowStates}-window` et myInferredNumArray = [1, 2, 3]; let mvNumArray: Array<number> = [1, 2, 3]: let myTupel: [number, string] = [1, "abcd"]; let unknownVar: unknown; let numberVar = 1; unknownVar = 3; // numberVar = unknownVar

// geht nicht, keine flow analysis if (typeof unknownVar === 'number')

numberVar = unknownVar // geht wegen narrowing function add(s1: string, s2: string): string; function add(n1: number, n2: number); number; function combineFunction(sn: number | string = ""

ns?: number): string { return String(sn)+String(ns || "");} function numberApplicator(numArray: number[], numFun: (prevRes: number, current: number) => number): number { return numArray.reduce(numFun); } interface PointList<T> { itemList:T[]; } // Generics const myCanvas = document.getElementById('main canvas") as HTMLCanvasElement;

Types/Interfaces: Interface verwenden, ausser man braucht Feature von Type (union, intersection, primitive) Strict Mode ("strict": true compilerOption in tsconfig.json): noImplicitAny, noImplicitThis, alwaysStrict, strict-BindCallApply, strictNullChecks, strictFunctionTypes,

strictPropertyInitialization, useUnknownInCatchVariables

4 Responsive Design

Flexible Layout: zusätzlicher Platz wird durch Verbreiterung der Elemente ausgefüllt, lässt sich ohne Media-Queries umsetzen, dynamisch, grössenadaptiv Responsive Layout: zusätzlicher Platz wird durch Um ordnung und zusätzliche Elemente / Spalten ausgefüllt,

braucht Media Queries, dynamisch, optimiert für untersch. Geräte / Bereiche von Display-Grössen, jedes der ayouts ist meist ein flexibles Layout

Graceful Degradation: App wird mit voller Funktionalität für moderne Browser geschrieben, dann werden Layers entfernt, damit es auch auf älteren Browsern funktioniert Progressive Enhancement: App wird nur mit Funktionen geschrieben, die alle Browser supporten, dann werden fortgeschrittenere Features hinzugefügt wie Layers Mobile First Layout: wie PE, technische Bedeutung: Base CSS ist für Mobile, grössere Displays werden extra hinzugefügt, Design Bedeutung: Wireframes für Mobile CSS Reset: reduziert Inconsistencies zwischen Browsern, löscht Browser Default CSS, viele Properties muss man danach erneut setzen, was den Code aufbläst CSS Normalization: Properties werden genormt über alle Browser, sie verhalten sich also alle gleich, sinnvolle Voreinstellungen wie Abstände bleiben erhalten Media Queries: spezifisches CSS für untersch. Medien

@media print and (min-color: 1) { ... } @media (min-width: 600px), (max-width: 800px) { ... } @media (orientation: landscape/portrait) {...} @supports not (display: grid) { div { float: right; } } @media screen and (hover: hover | none) { ... } @media (pointer: fine | coarse | none) { ... } @media (min-resolution: 300dpi) {...} link rel="stylesheet" href="big.css" media="(min-width: 62em)">

Media Query Operatoren: "and", ",", "not", "only" Einheiten: px (Pixel, wird durch Zooming verändert), rem (font size des root Elements), em (font size des parent element), % (relativ zur Grösse des parent element) Trigger Punkte: 480px / 30em (Smartphone), 768px / 48em (Tablet), 992px / 62em (Desktop)

Container Query: eigene Breakpoints pro Komponente

container { container-type: inline-size; } .card { background: lightblue: } @container (min-inline-size: 500px) { .card { background: lightgreen; } } container-2 { container: my-cont / inline-size; } @container my-cont (min-inline-size: 500px) { ... }

Query Units (bezogen auf Container Query): cqw (width), cah (height), cai (inline-size), cab (block-size). cgmin (min(cgi, cgb)), cgmax (max(cgi, cgb))

Viewport: unterbindet Intelligenz mobiler Browser, welche Seitengrösse überdimensioniert für Sites ohne Tag

<meta name="viewport" content="width=device-width,initial-scale=1">

position relative, negative margin-top, position absolute







Display: inline (margin l/r, padding, top, bottom, width, eight), block (margin, padding, overflow: scroll/hidden/ invisible), inline-block (margin, padding, width, height,

rertical-align: top) **Box sizing**: calc(100vw-5em) min(500px,100vw-5em) nax(400px,100vw-5em) clamp(400px,10vw-5em,500px)

Flexbox: Kinder eines Flex containers können flexibel angeordnet und ausgerichtet werden

container { display: flex|inline-flex; flex-direction: row|row-reverse|column|column-reverse; flex-wrap: nowrap|wrap|wrap-reverse;

flex-flow: direction wrap; gap: gap|row-gap column-gap; justify-content: flex-start|flex-end|center|spacebetween|space-around|space-evenly|start|end|left|right;

align-items: stretchlflex-startlflex-endlcenterlbaselinel first baselinellast baseline|start|end|self-start|self-end; align-content: flex-start|flex-end|center|spacebetween|space-around|space-evenly|stretch|start|end| baselinelfirst baselinellast baseline: }

item { order: 5: /* def. 0 */ flex-grow: 4: /* def. 0 */ flex-shrink: 3; /* def. 0 */

flex-basis: | auto; /* def. auto */

flex: none I grow shrink basis; align-self; autol flex-startlflex-endlcenter/baseline/stretch; }

Grid: 2-D Raster zur flexiblen Anordnung von Elementer

container { display: grid|inline-grid;

grid-template-columns: 50px auto repeat(5, 1fr): grid-template-rows: min-content 5px 1fr max-content; grid-template-areas: "header header" "main . sidebar "footer footer":

grid-template:noneltemplate-columns/template-rows: column-gap: 1px; row-gap: 1px; gap:<row> <column> justify-items: start | end | center | stretch; align-items: start | end | center | stretch;

iustify-content: startlend|center|stretch|space-around| space-between|space-evenly;

align-content: startlendlcenter/stretch/space-around/ space-between|space-evenly;

grid-auto-columns: 3px 4px ...; grid-auto-rows: 3px 4px ...; }

item { grid-area: headerl<rows>/<cols>/<rowe>/<cole> arid-column: <start> / <end> | <start> / span <value>: grid-row: <start> [/ <end>] | <start> / span <value>; justify-self: start | end | center | stretch;

align-self: start | end | center | stretch: }

5 Accessibility

Rechtlich: UNO-Behindertenrechtskonvention (UNO-BRK), gleichberechtigte Möglichkeiten im Zugang & Benutzung von Informations- & Kommunikationssystemen Häufigste Fehler: Text zu wenig Kontrast, kein alt Text, leere Links/Buttons/Doc-Sprache, fehlende Form Labels Farbenblindheit: prüfen mit Dev-Tools (Rendering),

Doppel-Codierung notwendig (Farbe & Form/Icon/Text), kritische Informationen nicht nur mit Text codieren, farbenblind-freundliche Farbpaletten wählen

Farbkontrast: prüfen mit Dev-Tools (Color Picker) & Lighthouse / Firefox Report, Zielgruppe erweitern mit WCAG Kontrast Level AA (5.7:1, 50+), AAA (15.9:1, 80+) Zoombarkeit: Zoom nicht unterbinden, Falsch: user-sca lable=0/no. maximum-scale=1

Animationen: sollten abstellbar sein, Verringerung der Ablenkung bei Konzentrationsstörung. Verhinderung der Auslösung von Epilepsie und Migräne

@media (prefers-reduced-motion: reduce) { ... } link rel="stylesheet" href="animations.css" media="(prefers-reduced-motion: no-preference)">

Tastatur-Bedienbarkeit: alle wichtigen Input-Elemente sind in der richtigen Reihenfolge fokussierbar, verboten: float, flex-direction: *-reverse, Standard Input Controls nutzen: a, button, input[type="..."], textarea, zusätzliche Elemente können mit "tabindex=0" fokussierbar gemacht werden, Elemente jeder Seite können mit "autofocus" versehen werden

Screenreader-Optimierung: keine Headings Level auslassen, semantische Elemente richtig nutzen, Skip-Links am Anfang der Site, lang Attribut korrekt setzen

ARIA Attribute: lösen Accessibility Issues, die nicht mit nativem HTML gelöst werden können, heissen jeweils aria-*, wichtigste: hidden, label, expanded, current, labelledby, controls, haspopup, invalid, live, required

Tabellen: Heading für Rows & Columns mit ausstatten, alternativ Allgemeines: alt Tag bei Bildern, leer wenn dekorativ, logische Reihenfolge auch ohne CSS, semantische Struktur, Multimedia: 2-Sinne Prinzip, Anzeige-Art kann angepasst werden (Textgrösse, Animationen, Zeitlimits), Über schriften/Form Labels/Linktexte sind verständlich, auf Formatwechsel (z.B. PDF) wird hingewiesen, konsistent Navigation, valides HTML, kompatibel mit versch, I/O Geräten, Hilfestellung bei Interaktionen, PDFs auf PDF/ UA Konformität validieren, keine Tabellen für Lavout

6 Security

Stored XSS: Server kann so manipuliert werden, dass JS Code an die Browser der Opfer ausgeliefert & ausgeführt wird, Massnahmen: User-Input muss beim Output encodet werden, z.B. mit 3x {} in Handlebars, oder User-Input wird sanitized, z.B. mit xss oder dompurify Library JS Injection: dem Server kann JS Code gesendet werden, welcher ihn im Code mit eval()/setTimeout()/setInterval()/Function() ausführt, Massnahmen: stattdessen parseXXX()/JSON.parse() verwenden, globale Scopes & Variablen reduzieren, rechen-intensive Tasks mit childprocess.spawn auslagern, Node kein Root-Prozess Insecure direct object references: durch Manipulation der URL lassen sich sensible Daten ohne Authentifizierung oder Autorisierung aufrufen, Massnahmen: bei allei Sites mit sensiblen Daten sicherstellen, dass der User eingeloggt und berechtigt ist

const u = findUser(req.body.name); rea.session.user id = u.id: // if user exists & PW correct

if user.isAuthorized(reg.session.user_id, reg.url) { next(); } else { res.send(401, 'not authorized'); } } app.all("/api/*", login, function(reg, res, next) { next(); })

Cross Site Request Forgery: Angreifer bringt Nutzer mi gültiger Session dazu, auf einer Site ein gefälschtes Formular an die richtige Site zu submitten, Massnahmen: Formular-Submission-Requests (GET/POST) müssen geprüft werden, ob sie von einem vom Server frisch aus gelieferten Formular stammen (mit CSRF Token)

const csrf = require("tiny-csurf"); app.use(csrf({ cookie: true }));

function login(reg, res, next) {

router.route("/form").get(function(reg, res, next) { res.render("form", { csrfToken; reg.csrfToken() }); }) .post(function(reg, res, next) {

f(checkAndRecordUnseenCSRFToken(reg.body. csrf)) res.send("qood")} else { res.send("bad") } });

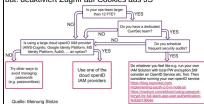
Replay attacks: z.B. bei Ouiz wird mit Formular Aufgabe und User-Input-Lösung submitted, wenn richtig gibt es einen Punkt, das kann replayed werden, sollte aber nicht gehen, da der User dann unrechtmässig Punkte erhält, Massnahmen: CSRF Token 1x gülitg, bereits gelöste Auf gaben speichern, letzte Aufgabe in Session speichern Cryptographic failures: Crypto schlecht eingesetzt. Massnahmen: https nutzen, keine geheimen Infos in Ouery-Params (Leaks in Server Logs oder Browser History), Authentication Service nutzen

Identification & Authentication failures: z.B. ist Session Timeout zu lang, User bleibt auf öffentlichem PC eingeloggt, Massnahme: Session Timeout sinnvoll setzen CORS Header: ermöglicht Laden von Ressourcen (Bilder, Scripts, usw.) von anderem Ursprungsort als eigene Origin, man beschränkt, welche Websites auf seine Ressourcen zugreifen dürfen

Access-Control-Allow-Origin: https://test.ch | * Access-Control-Allow-Methods: GET. POST

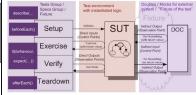
CSP Header: Content Security Policy, Beispiel schränkt Communikation auf Source Domain/api.ch ein, externe & Inline-Scripts sind verboten, aktivieren mit unsafe-inline Content-Security-Policy: default-src 'self' *.api.ch

Weiteres: DNS Prefetching & Client Caching deaktivieren, Public Key Pinning Headers, Strict-Transport-Securi y Header (HSTS) setzen, X-XSS-Protection Header, Safe-Regex verwenden (DDoS), Parameter pollution Cookies: Secure Attribut: Cookie wird nur über HTTPS versendet, nicht HTTP (ausser localhost), HttpOnly Attribut: deaktiviert Zugriff auf Cookies aus JS



7 Testina

Unit Test: einzelne Klassen/Module (meist ein File) werden getestet, einfache Automation, Herausforderungen: Units isolieren, asynchrone Operationen, Testdaten-Generiung (z.B. Fuzzing)



Integration Test: Zusammenspiel von 2 oder mehr Units wird getestet, Automation meist möglich, Herausforderungen: gleiche wie bei Unit Tests. Simulation Browser & Benutzerinteraktion, Test mit Datenbank

End-to-End Test: Integration Test über alle Laver Anderes: Smoke, Static (TypeScript), Regression, Load Performance, Endurance, Chaos, Security, Usability

Test-Runner: Rahmen, der Tests entgegennimmt, ausführt und Resultate anzeigt, z.B. Mocha, Cypress Assertion Library: gut lesbare Annahmen schreiben, die

nicht verletzt werden sollten, z.B. Assert, Chai Mocking Library: Separierung von Units, Erstellung von Test-Doubles insb. Mocks, z.B. Proxyguire, Sinon, is

const { expect } = require('chai'); describe('Array', function() {

describe('#indexOf()', function() {

beforeEach(function() { this.testArray = [1, 2, 3]; }); it('should return -1 when the value is not present',

expect(this.testArray.indexOf(4)).to.equal(-1); }); }); })

Test Double Pattern: Bei Unit Tests hat man zum Teil ein Dependent On Component (DOC) an der Unit, mit Si non kann man einen vorbestimmten Input (Stub/Fake) fürs SUT und überprüfbaren Output (Spy/Mock) liefern Eigenschaften guter Unit Tests: Output statt Internes testen, jeder Test hat voraussagbare & deklarative Struktur, nur eine Aktion und der Output davon testen, keine Daten mit anderen Tests teilen, alles Nötige zum Verstehen der Absicht muss im Test sein

Unit Test Smells: Hard-to-Test Code, viele Bugs bei formalem Testing & Produktion, Tests failen wenn SUTs so

verändert werden, dass die Tests nicht beeinflusst werden sollten, Mal-so-mal-so Tests, Conditionals in Tests. Test-Code Duplikate, langsame Tests, schwierig zu verstehende Tests, Test-Logik im Prod-Code, schwer zu sagen welches Assert im Test failed, keine Tests

8 Internationalization

Internationalisierung: Software-Entwicklungs-Methode damit alle Benutzerausgaben per Spracheinstellung austauschbar und nicht hard-coded sind

Lokalisierung: Inhalt den sprachlichen & kulturellen Eigenheiten bestimmter Zielkulturen/Länder anpassen Übersetzung: Inhalt übersetzen, Arbeit der Übersetzer Locale: String, der Region (Sprache & Land) bestimmt Unterschiede Sprachregionen (nicht halbautomatisch anpassbar): kulturelle Unterschiede (Bilder, Symbole, Farben), regulatorische Anforderungen (Daten-/Verbraucherschutz), sprachliche Feinheiten (Dialekte, Idiome, Redewendungen), UX und Usability (Navigation, Benutzerführung, Leserichtung), Markt- & zielgruppenspezifische Anpassungen (Produktangebote, Werbung, Marketing), kulturelle Sensibilitäten (politische & soziale Themen, Feiertage), Titel & Anreden, anderes Konzept als /orname/Nachname. Schreibweise von Adressen & Telefon-Nummern, Satzzeichen, Masseinheiten (Umrechnung), Kalender, Sounds, Lavout, Steuern

const rn = new Intl.DisplayNames(['de'], {type: 'region'}); console.log(regionNames.of('CH'));

const cg = new Intl.Collator('de').compare('ä', 'z'); c.log({cg}); // negativ, 'ä' kommt nach 'z'

const d = new Date(Date,UTC(2022, 4, 17, 13, 30, 0)); c.log(new Intl.DateTimeFormat('en-US').format(d));

const rtf_english = new Intl.RelativeTimeFormat("en", { localeMatcher: "best fit", // oder: "lookup"

numeric: "always", // oder: "auto" style: "long" }); // oder: "short" / "narrow"

c.log(rtf english.format(-1, 'day')); const m = 123456.789

c.log(new Intl.NumberFormat('de-CH').format(n)); const colors = ['red', 'blue', 'pink', 'orange']:

const formatter = new Intl.ListFormat('en', { style: 'long', type: 'conjunction' });

console.log(formatter.format(colors));

const w = new Intl.PluralRules('en-ÚS', {type: "ordinal"}); |Modularisierung, Funktionalitäten wiederverwenden console.log(w.select(1)); // "one" const deCHLocale = new Intl.Locale("de-CH");

9 Animation

Most properties that accept a length, number, color, or the function calc() can be animated. Most properties that take a keyword or other discrete values, like url(), can't.

ransition-property: name z.B. color | all | none; transition-duration: 3s | 300ms;

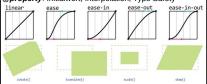
transition-timing-function: linear | ease-in | ease-out | ease-in-out:

ransition-delay: 0.3s | 300ms;

transition: property duration [timing-function delay], [...] translateX(3px) | translateY(-3px) | translate(x, y) | $scaleX(1.3) \mid scaleY(0.7) \mid scale(x, y) \mid$ $skewX(3deg) \mid skewY(-3deg) \mid skew(x, y) \mid none [...]$

ransform-origin: %, px, left, right, top, bottom, center;

@property: Initialwert, Interpolation, Type Safety



@keyframes wiggleAnimation { transform: skewX(9deg); } { transform: skewX(-9deg); } % { transform: skewX(0deg); } } wiggle { animation-name: wiggleAnimation; animation-duration: 1s: }

LO User Experience

Don't listen to users: Schauen was User machen, nicht vas sie sagen, Kunden fragen, User nicht, Umfragen **Problem Space**: User analysieren, beschreiben, haben Redürfnisse und Probleme

Solution Space: Designer, Vision, Storyboard, Prototyp Problem / Future Scenario: User, Problem Beschreibung, Kontext, Trigger, Schritte, Lösung / Fail Gute UI Ausschilderung: Wo bin ich? (Titel, Breadcrums) Wo kann ich hin? Was ist passiert? Kofferraum Concept Model: Benutzer, Ressource und Beziehungen Site Map: Seitenhierarchie (Baum) wie in der Navigation Card Sort: Cards mit allen Seiten erstellen, Cars gruppieren & Gruppen benennen (Open), Gruppennamen vaidieren (Closed). 1 Hierarchiestufe

Tree Testing: Site Map aufnehmen, Szenarios zur Erreichung von Zielen stellen. Wireframe: Screen-Skizze Screen Flow: Wireframe-Abfolge, zeigt User-Szenario Gute Test-Szenarien: plausible Ziele, Kontext, Trigger, Skills, keine Schritte sagen, neutrale Personen, Konzept nicht Usability testen, Pre- & Post-Umfragen, think aloud Usability Kriterien Nielsen: Sichtbarkeit des System-Status, enger Bezug zwischen System und realer Welt, Nutzerkontrolle und Freiheit, Konsistenz & Konformität mit Standards, Fehler-Vorbeugung, besser Sichtbarkeit als sich-erinnern-müssen. Flexibilität und Nutzungseffizienz, Ästhetik und minimalistischer Aufbau, Nutzern helfen. Fehler zu bemerken, zu diagnostizieren und zu beheben, Hilfe und Dokumentation

Messen: Effektivität – User können Ziele erreichen. Effizienz – angemessener Aufwand zur Zielerreichung, Zufriedenheit: positive Einstellung gegenüber System

11 Web DevOps

CSS Präprozessoren: SWE Prinzipien, weniger C&P,

\$purple-navy: #635380: /* Variablen in Sass */ ul { &:hover { color: \$purple-navy; }} /* auch in SCSS @import 'importvars' /* \$var nutzen, importvars.css */ @use 'usevars' /* \$usevars.var nutzen */ @mixin sth(\$var: 1px) { foo: bar; abc: \$var; } box { @include sth(2px) } %icon { background-color: green; } /* %=abstract */

error-icon { @extend %icon; color: blue; }

PostCSS: Framework zur CSS-Tool-Entwicklung, Parser → API → Plugins → Stringifyer, z.B. AutoPrefixer für browser-specific CSS, Minifier, Polyfill (modern → alt) Frameworks: Bootstrap, Tailwind, Material UI ransform: rotate(-3deg) | rotateX(-3deg) | rotateY(-3deg) | Build Tools: komplexe grosse Projekte, häufige Aktualisierungen, Teamarbeit, Minimierung, Transpilierung, Modulbündelung, Automatisierung, Linting, Sourcemaps Weitere Kategorien: IDE, Package Manager, Code Control, Code Formatodoter, Linter, (E2E) Testing Tool, CI Service, SSR, DB, CMS, Logging, Cloud Service

12 Undo

Vorteil ggü. aktiven Notifikationsdialogen: Fehler einfach korrigieren, unnötige Dialoge vermeiden, weniger Angst, experimentieren geht, Redo ist auch wichtig Wichtige Design-Erwägungen: Kontext (App, Dokument, Feld), Granularität (Buchstabe(nsequenz)), Operationen (Edit, Select, Resize, Print?, Send?)