Partnerarbeit von lroellin und dthoma

1 REST

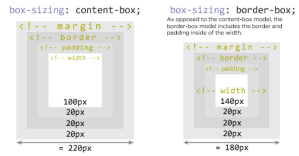
URL Regeln Ressourcen werden immer mit einer URL identifiziert (Resource Oriented Architecture **ROA**). Ressourcen in Mehrzahl, Query-Parameter nur für Algorithmen oder Filter (`/orders?state=delay`). Hat man verlinkte Ressourcen, so kann man sie entweder direkt in der Antwort mitgeben, oder die URL zum Abruf dieser Ressource zurückgeben.
HTTP-Methoden GET: abrufen, Accept-Header definiert Repräsentation. Status-Codes beachten. **POST**: neue Ressource erzeugen. `Content-Type` Header angeben. Wenn die Ressource erzeugt wird, im Response-Header. `Location` die URI angeben. Kein Cache. **PUT** aktualisiert eine Ressource, oder erzeugt sie falls noch nicht vorhanden. Idempotent! Kein Cache. **DELETE** löscht eine Ressource. Kein Cache. **OPTIONS** gibt an, wie die Ressource verwendet werden darf, z.B. welche Methoden erlaubt sind. **HEAD** genau gleich wie GET, aber ohne die eigentliche Ressource. Mit Cache. Header-Informationen und Status-codes sind relevant. **PATCH** wird für partielles Updaten genutzt. **Repräsentation** Wie die Ressource schlussendlich repräsentiert werden soll (JSON, XML, Excel-Tabelle, ...) entscheidet der `Accept-Request-Header`
HATEOEAS Hypermedia as the engine of application state. "Browsable API", selbstbeschreibend, technisch sehr schwer umsetzbar.
Versionierung Oftmals ein Knackpunkt. In der Realität oft über URL, auch wenn das ROA widerspricht. Andere Varianten wären der Media-Type im Accept. Am besten: keine Versionierung
Best Practices Use nouns but no verbs. Use plural nouns. Use HTTP status codes. Respect the meaning of the HTTP methods: GET method should not alter the state. PUT is idempotent, POST is not idempotent.

2 Responsive Design

Device Pixel CSS-Pixel sind nicht Device-Pixel. Smartphones haben unterschiedliche Pixel Ratios.
Browser nicht auf dem selben Stand Zwei Vorgehensweisen. **Graceful Degradation** Alle modernen Features nutzen. Bei älteren Brwosern Polyfills nutzen, sinnvolle Alternative geben oder auf Problem hinweisen. **Progressive Enhancement** für alle Browser zugänglich machen. Mit Grundfunktionalität starten (kein JS, keine Media Queries), dann mit zusätzlichem CSS und JS Zusatzfunktionalität bieten.
Support abfragen Mit `supports (flex-wrap: wrap) {}` kann abgefragt werden, ob ein Browser ein Feature unterstützt. Es gibt aber auch den Modernizr.
Responsive Images `src` Fallback, `srcset` Liste von Bildern mit zusätzlichen Informationen wie Grösse oder Pixeldichte, `sizes` gibt an, wie gross das Bild im Layout sein sollte (`[media query] [length], [media query] [length]`). Browser wählt aus!

```
<img src="image-src.png" srcset="300.png 300w,
  600.png 600w"sizes="(max-width: 320px) 49vw, 100vw">
<img src="image-src.png"  srcset="image-1x.png 1x,
  image-2x.png 2x, image-3x.png 3x">
```

2.1 CSS Box-Modell



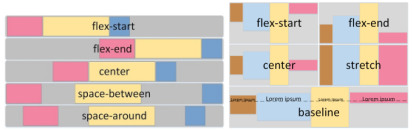
vH: % der View-Height (vW Width, 20%: % vom Parent, `calc(100vW - 5em)` berechnet. `margin` kann auch negativ sein.

2.2 Inline/Block/Inline-Block

Inline wie `span a`, erlaubt left/right Margin/Padding, aber nicht top/bottom. Ignoriert width/height. Erlaubt andere Elemente auf der gleichen Zeile. White-Space zwischen Inline-Elementen wird dargestellt (Space). Forcieren: `display: inline`
Block wie `h1 div form p` , erlaubt Margin/Pading, jedes Element auf einer neuen "Zeile" (Umbruch), erlaubt Text-Inhalte zu scrollen, clippen (`overflow: scroll | hidden | invisible` Forcieren: `display: block`
Inline Block wie Inline-Flex, Inline-Table erlaubt Margin/Pading, erlaubt andere Elemente auf der gleichen Zeile mit Alignment (`vertical-align: top`), erlaubt width/height, White-Space zwischen Inline-Block-Elementen wird dargestellt (Space) Forcieren: `display: inline-block`

2.3 Flexbox

Container: `display: flex` / `display: flex-inline`. Alle Flex-Items verhalten sich wie `inline-block`.



Main: `justify-content` (Bild links) **Cross:** `align-items` auf Container. Default: `flex-grow: stretch` (Bild rechts).

Bei mehreren Zeilen `align-content:` mit allen properties wie `justify-content:`.
Ausrichtung: Kann mit `flex-direction: row` (links-rechts) oder `column` (oben-unten) und mit `-reverse` auch die Reihenfolge geändert werden.
Umbruch: `flex-wrap: nowrap` (single line), `wrap` (wrap around additional lines), `wrap-reverse`. `flex-flow: [flex-direction] [flex-wrap]` kombiniert direction mit wrap.
Grösse: `flex-grow: 1`: entspricht dem Verhältnis, wie der "leere Platz" verteilt wird (default `0`, nicht grösser werden). `flex-shrink: 1`: entspricht dem Verhältnis, wie die Elemente kleiner werden, wenn zuwenig Platz (default `0`). `flex-basis: 100px | 10% | auto`: gewünschte Start-Grösse des Elements (auch über `width/height` definierbar). Kurzschreibweise `flex: [flex-grow] [flex-shrink] [flex-basis]`
Item: Wenn ein Element in der falschen Reihenfolge ist: `order` default `0`. "Zuerst" werden die Elemente mit der kleinsten Order angezeigt. **Negative Elemente sind kleiner als 0 und nicht "vom Ende her" gezählt.** Dasselbe wie `align-items` mit `align-self` auf dem einzelnen Item.

2.4 Grid

Container:

```
grid-template-columns: 1fr auto 20%;
grid-template-rows: repeat(4, 1fr);
```

fr: Fraction of available Space, auch dezimal. Können nicht schmaler als das längste Wort werden. Overflow wird vermieden.
max-content/min-content grösst-/kleinstmöglich.
auto maximale gewünschte Breite der Grid-Elemente in der Spalte.
minmax(min, max): Wert zwischen `min` und `max` wird sichergestellt. `minmax(auto, 1fr)`: soviel wie vom Inhalt vorgegeben wachsen mit `fr`.
repeat(times: int, measure) Wiederholung der measure, `grid-template-columns: 10em repeat(3, 1fr 2fr)` `10em` => 8 Spalten: 10em, 3x (1fr/2fr) abwechselnd, 10em.
Item:

```
grid-column-start: 1; /* mit 1 indexiert */
grid-column-end: span 2; /* 2 ab -start */
grid-row-start: 2;
gird-row-end: -2; /* vis a vis gezählt */
grid-area: 2 / 1 / -2 / span 2; /* Kurzschreibweise */
/* [row-start]/[column-start]/[row-end]/[column-end] */
order: -2; /* default 0 */
```

template-areas

```
grid-template-areas: "aa bb bb" /* Container */
                     "aa . dd";
grid-area: aa /* Item */
```

Element Alignment (Y): `align-items: stretch | start | center | end` (Default: `stretch`). Selber: `align-self`. Element Justification (X): `justify-items: stretch | start | center | end` (Default: `stretch`). Selber: `justify-self`.

2.5 Media Queries

```
@media ([width|min-width|max-width]: 375px) {}
@media ([height|min-height|max-height]: 667px) {}
@media ([device-width|min-device-width|
    max-device-width]: 375px) {}
@media ([device-height|min-device-height|
    max-device-height]: 667px) {}
```

`@media (orientation: landscape)`, oder `min-resolution: 300dpi`. Können auch kombiniert werden: `@media (min-width: 20em) and (max-width: 30em)` oder auch `@media only screen` (oder print).
Breakpoints: `480px/30em`: Smartphones, `768px/48em`: Tablets, `992px/62em`: Desktops
Style Tag: `<link rel="stylesheet" href="style.css" media="(min-width: 30em)">`
Bei Texten `max-width: 60em` verwenden gegen überlange Zeilen. Media-Queries können in JS abgefragt werden: `if (window.matchMedia("(minWidth: 35em)").matches)`

2.6 Viewport

Bei einer responsiven Seite setzt man den Viewport, damit der Browser nicht "intelligent" versucht zu zoomen. Ohne, greifen die Media Queries nicht. `<meta name="viewport" content="width=device-width, initial-scale=1">`.
`width=device-width`: die Seite soll so breit sein, wie das Gerät.
`initial-scale: 1` zoome zu Beginn so, dass es 100% entspricht.

3 Sass/SCSS

$ Variablen, & Parent-Selektor, % Placeholder (Abstract), @ Steuerzeichen (`@if`, `@else`), #{} Interpolation (p.#{$name}), `/* */` und `//` Kommentare.
Variablen:

```
$colorMain: #abcdef;
a { color: $colorMain }
```

Nesting:

```
nav {
  ul { display: block }
  li { display: inline-block }
  > a { padding: 6px }
} /* => */
nav ul { display: block }
nav li { display: inline-block }
nav > a { padding: 6px }
```

Parent Selector & wird durch Parent-Selektor ersetzt

```
.headline {
  margin-top: 5em
  .sidebar & { margin-top: 2em }
  &:hover { color: darkred}
} /* => */
.headline { margin-top: 5em }
.sidebar .headline { margin-top: 2em }
.headline:hover { color: darkred }
```

Partials/Imports müssen mit einem Underscore beginnen (_), werden nicht in separates CSS übersetzt. Bei `@import` muss die Dateiendung und der _ nicht angegeben werden.

```
// file: _constants
$brand-color: #abcdef
$width: 960px
// anderes File
@import 'constants'
h1 { color: $brand-color }
body { width: $width }
```

3.1 Mixins

`@mixin` definiert neues Mixin, `@include` lädt das Mixin. Auch mit Parameter (und Default).

```
@mixin visuallyhidden($border: 1em) {
  border: $border
  height: 1px /* ... /*
}
.element {
  @include visuallyhidden(1rem)
}
```

`@content` wird mit dem Content von `@include {}` abgefüllt. Auch mit Parametern nutzbar.

```
@mixin breakpoint($size) {
  @media screen and (min-width: $size) { @content }
}
@include breakpoint(30em) { ... }
```

`@content` wiederum kann beliebiges SCSS sein, inkl. Includes. Auch & funktioniert richtig:

```
@mixin hover() { &:hover { background: red } }
.box { @include hover() } /* => */
.box:hover { background: red }
```

Extend/Inheritance

```
.icon { margin: 0.5em }
.error-icon { @extend .icon; background: red }
.info-icon { @extend .icon; background: blue }
/* ==> */
.icon, .error-icon, .info-icon { margin: 0.5em }
.error-icon { background: red }
.info-icon { background: blue }
```

Mit % kann Abstract Class genutzt werden.

```
%icon { margin: 0.5em }
.error-icon { @extend %icon; background: red }
.info-icon { @extend %icon; background: blue }
/* ==> */
.icon, .error-icon, .info-icon { margin: 0.5em }
.error-icon { background: red }
.info-icon { background: blue }
```

Einheiten / Rechnen Numbers (`1, 1.2, 1px`), Strings mit oder ohne `""`, Colors, Booleans, Nulls, Listen (mit Komma oder Leerzeichen, `$list = 1.5em 1em 0 2em`), Maps: `(k1: v1, k2: v2)`. Beispiele: `2px + 2px = 4px, 2px/1px = 2, 2px + 1mm = 5.78px, 10 + px = "10px"` aber Achtung: `10px * 10px = 100px*px = Error`
Loops/Verzweigungen

```
$width = 2px;
$breakpoints: 30em 46em
@each $point in $breakpoints {
  @media all and (max-width: $point) {
    body {
      @if $point > 40 em { width: $width }
      @else { width: $width * 2 }
}}} /* => */
@media all and (max-width: 30em) { /* else */
  body { width: 4px }
}
@media all and (max-width: 46em) { /* if */
  body { width: 2px }
}
```

Funktionen

```
@function properZero($param) {
  @if (komplizierte Rechnung mit $param) {
    return $param / komplizierte Rechnung
  } @else { return $param }
}
body { width: properZero(0px) }
```

3.2 CSS-Variablen (ohne SASS)

Beginnen mit `--`. Da alle Properties sich dem DOM nach vererben, definiert man Custom Properties typischerweise auf dem Body, und nutzt sie dann später mit `var()`. Anders als Sass-Variablen, können sie zur Laufzeit abgefragt und geändert werden.

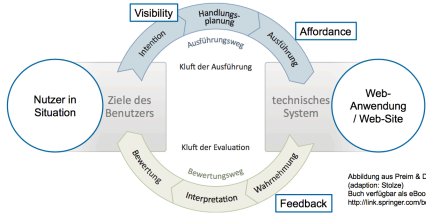
```
body { --main-bg-color: red }
div { background-color: var(--main-bg-color) }
/* mit Default Wert */
div { background-color: var(--main-bg-color, red) }
/* auch mit calc */
body { font-size: calc(2 * var(--base-font-size)) }
/* übersteuern */
@media screen and (min-width: 560px) {
  body { --main-bg-color: blue }
}
```

In JS abfragen: `element.style.getPropertyValue("--my-var")`, setzen: `element.style.setProperty("--my-var", jsVar + 4)`

4 UCD

Accessibility ist nicht nur "mit Screenreader testen"!
Regeln: Jedem Bild und Animation Beschreibung zuordnen (`alt`), keine Informationen ausschliesslich mit Farbe darstellen, Vorder-/Hintergrund auch bei weniger Farbe/Konstrast deutlich unterscheidbar machen, Skalierung der Schrift (`em`/`rem`/`%` statt `px`), Tastaturbedienung, Altersgerecht: leichter ablenkbar (keine Animation), Fonts unter 11pt kaum lesbar, Targets nicht zu klein, im DOM Navigation zuerst, Bereiche gruppieren, semantic Markup nutzen, komplementäre Farben vermeiden.
Accessibility Tree Der "Accessibility Tree" ist die von Screen-Readern genutzte, vereinfachte und mit Zusatzinformationen ausgestattete Repräsentation des HTMLs.

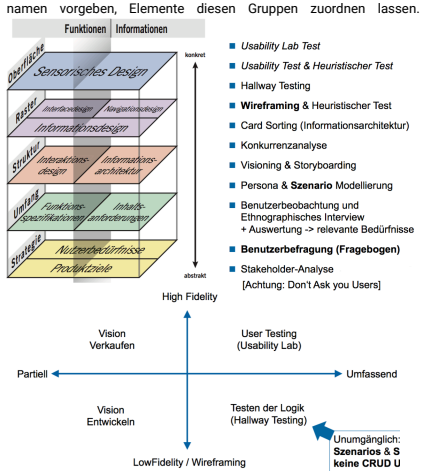
4.1 Norman



Kontrast 15.9:1 für 80+, 5.7:1 für 50+.
Affordance wie kann ich etwas bedienen? Türen: Klinke, Hebel. Links blau unterstreichen. Logo oben links/rechts führt zur Homepage. Grauer Hintergrund-Text ist Label oder Instruktion. Login/Nutzerinformation ist oben rechts. Navigation-Menü ist oben. Such-Felder haben Lupe. Buttons sollen wie Buttons aussehen. **Instruktionstexte sind Symptom für schlechte Affordance**.

4.2 Card Sorting/Garrett/Wireframe

Card Sorting (1) Content Repository erstellen, (2) **Open Card Sort:** 5+ Personen aus der Zielgruppe, Elemente in disjunkte Gruppen ordnen, (3) gute Gruppennamen identifizieren, (4) **Closed Card Sort:** validieren: 5+ neue Personen, Gruppennamen vorgeben, Elemente diesen Gruppen zuordnen lassen.



4.3 Tree Testing

Zugeklappten Menü-Tree vorgeben, Nutzer Aufgabe geben "Finde X", sehen ob er es findet, und wo er sich durchhangelt.

4.4 Kriterien

Tastaturbedienbarkeit, semantische Struktur, Multimedia/2-Sinne Prinzip, Kontrast, Syntax/Kompatibilität, Hilfestellung bei Interaktionen

5 Security

5.1 XSS Cross Site-Scripting

In einem Formular angegebene Daten werden an andere Nutzer ohne Escaping ausgeliefert. Encode for ... **HTML Body** < => `<`, **HTML Attributes** nicht-alphanumerisches: &#xHH-Format. **CSS** CSS Hex Encoding `\HHHHH`, **JS** nicht-alphanumerisches `\uXXXX`, **URI** `encodeURI`, **URL Parameter** `encodeURIComponent()`. Ferner CSP setzen und httpOnly (nicht in JS auslesbar) + secure auf dem Cookie.

5.2 JS Injection

Nie `eval` verwenden! Möchte man z.B. einen Zahlenstring in einem Request mit `eval` in eine Zahl verwandeln, so könnte ein Angreifer dort beliebiges JS platzieren und damit den Server übernehmen.

5.3 Broken Authentication and Session Management

keine PW über HTTP, kurze Session Timeouts, keine ungehashten PWs

5.4 Insecure Direct Object References

Über eine manipulierte URL auf Daten zugreifen, für die man keine Berechtigung hat. Gegenmassnahme: sicherstellen, dass der Benutzer eingeloggt ist (Session-Token).

5.5 CSRF: Cross Site Request Forgery

Bei einer Seite angemeldet sein, und eine fremde Seite sendet in meinem Namen Daten an die erste Seite.

```
app.use(express.csrf())
app.use(function(req, res, next) {
 res.locals.csrftoken = req.csrftoken()
}

<input type="hidden" name="_csrf"
  value="{{csrftoken}}">}
```

5.6 Security by Obscurity

Powered By ausschalten, Session Cookie Namen generisch machen.

6 OO

JavaScript kennt kein Overloading! **Dieser Konstruktor (Factory-Methode) kann mit oder ohne `new` aufgerufen werden.**

6.1 this

```
function newCounterWithInc(init) {
  const o = {}
  o.count = init
  o.inc = function(delta) { this.count += delta }
  return o
}
const c = newCounterWithInc(7)
c.inc(5)
```

"Freie" Funktionen können mit `call/apply`, ferner `bind`, das `this` gesetzt bekommen.

```
function globalCount(delta) { this.count += delta }
const o = {count: 1}
globalCount.call(o, 3)
```

Unterschied `call/apply` bei **c**all gibt man die Argumente an die eigentliche Funktion mit **C**ommas weiter, bei **apply** mit einem **A**rray.
Mit `bind` erhält man eine neue Funktion, deren `this` dauerhaft gebunden ist.

```
const oFn = globalCount.bind(o)
oFn(3)
```

6.2 ES5 Konstruktor

Konstruktoren sind normale Funktion. Nach Konvention grossgeschrieben. Müssen mit `new` genutzt werden

```
function Counter(cInit) {
  this.cont = cInit
  this.inc = function (delta) { this.count += delta }
}
const c = new Counter(12)
```

6.3 Prototypes

JS verwendet das System der Prototypen. Dabei gibt es eine Prototype Chain, entlang derer die gewünschte Methode/Property gesucht wird. Wenn sie gefunden wird, wird `this` auf das ursprüngliche Objekt gesetzt und die Methode aufgerufen. Man kann auch nachträglich bei einem früheren Prototyp eine Funktion anhängen; diese wird auch dann gefunden wenn sie bei der Konstruktion des neuen Objekts noch nicht bekannt war. Jedes mittels Konstruktor erstelltes Objekt hat einen Pointer auf das Prototyp-Objekt des Konstruktors (`.__proto__`
Mit den Prototypen kann eine mehrstufige Vererbung erzielt werden. Wird heute nicht mehr empfohlen, aber die ES2015 Syntax wird entsprechend übersetzt.

```
function Shape( ) {}
Shape.prototype.getArea = function() {return 0;}
function Rect(width, length) {
  const r = {}
  r._width = width; r._length = length;
  r.getArea = function() {return r._width * r._length;};
  return r;
```

```
}
Rect.prototype = Object.create(Shape.prototype);
function Square(edgeLength) {
  const n = Number(edgeLength);
  // Nicht mit new aufgerufen
  if (this === window) { return new Square(n); }
  else { return new Rect(n, n); }
}
Square.prototype = Object.create(Rect.prototype);
```

Mit new oder ohne aufgerufen? wenn `this === global` (im Browser `window`) ist, wurde es *OHNE* new aufgerufen.

6.4 ES6 Verbesserungen

Richtige Klassensyntax und Properties mit Getter/Setter

```
class Counter {
  constructor({start = 0, step = 1}) = {}) {
    this._count = start
    this._step = step
  }
  get count() { return this._count }
  set count(newCount) { myCount = newCount }
  inc(step = this._step) { this._count += step }
  dec(step = this._step) { this._count -= step }
}
class DoubleCounter extends Counter {
  constructor({start = 0, step = 1} = {}) {
    super({start, step})
    this._step = this._step * 2
  }
}
const dc = new DoubleCounter({step : 4})
dc.inc()
```

7 Funktionales JS/Array Funktionen

```
// Filter: a.Filter(fn(element, [index], [array])
  : boolean)
function getNumberArray(i) {
  return i.filter(e => typeof e === "number)
}
// Map: a.map(fn(element, [index], [array])
  : modifiedE)
function incrementAllNumbers(i, increment) {
    return i.map(e => e + increment)
}
// Reduce: a.reduce(
  fn(acc, element, [index], [array]): result,
  startResult)
function sumAllNumbers(i) {
  return i.reduce((result, e) => (result + e), 0)
}
// Filter Duplicates
toRemove.reduce((acc, element) => acc.includes(element)
  ? acc : acc.concat([element]), []).sort();
```

`Object.keys(object)`: alle Attribut-Namen des Objekts, `Object.values(object)`: alle Attribut-Werte des Objekts, `Arrays.every(fn(e): boolean)`: prüfen, ob alle Elemente eines Arrays eine Funktion erfüllen.

```
// Welche Personen praktizieren dieselben Hobbies
data.filter(e => e.indexOf(hobby) >= 0)
    .map(e => e.name);
data.reduce((result, item) => {
 const key = item.hobbies.sort().join(",")
 result[key]) ? result[key].push(item.name)
              : result[key] = [item.name]
});
// Palindrom (maoam)
data === data.reverse()
// Filter Duplicates
data.filter((e, i, a) => a.indexOf(e) === i)
```

8 TypeScript

Achtung: Schreibt man eine Klasse, die von einer (in der Datei) später definierten Klasse erbt, gibt es einen Compiler-Fehler. Dasselbe mit Funktionen, da der Compiler sie zu diesem Zeitpunkt noch nicht kennt.

8.1 Variablen

Benötigen keine Typendeklaration, dann wird der Typ inferiert. Man kann sie explizit als any deklarieren, dann (1) können beliebige Variablen zugewiesen werden und (2) **Variable darf beliebigen anderen Variablen zugewiesen werden**. Um globale-Variablen aus nicht TS-Files zu nutzen, deklariert man sie mit `declare let SOMEGLOBALVAR`. Ist ein Variablentyp einmal inferiert oder festgelegt, kann man andere Typen nicht mehr zuweisen:

```
let inferredNumVar = 1; inferredNumVar = "hi" // NOK
let numVar : number = 1; numVar = "hi" // NOK
```

Möchte man eine Variable nur deklarieren, nutzt man `declare let foo: string`.

8.2 Komplexe Typen

Bei Arrays wird der Typ inferiert.

```
let infNumArray = [1, 2, 3]
let numArray: number[] = [1, 2, 3]
```

Bei Tupeln ist dies nicht möglich

```
let notInfTupel = [1, 'abcd']
let tupel: [number, string] = [1, 'abcd']
```

Enums können wie Basistypen genutzt werden

```
enum Color { Red, Green, Blue }
let c: Color = Color.Green
let colorTupel: [Color, number] = [Color.Green, 1]
```

8.3 Funktionen

Hier auch mit Overloading! Und optionalen Parametern

```
function add(s1: string, s2: string): string
function add(n1: number, n2: number): number
function add(n1, n2) { return n1 + n2 }
function combineFunction(sn: number | string = "",
  ns?: number): string {
  return sn.toString() + (ns || "").toString()
}
```

Auch Funktionen können als Parameter deklariert werden

```
function numberApplicator(numArray: number[],
  numFun: (prevResult: number, current: number)
  => number) :number {
  return numArray.reduce(numFun)
}
numberApplicator([1, 2, 3, 4], add)
```

8.4 Klassen

Erweiterte ES6 Syntax inkl. Properties und `private/readonly`. `private` kann überschrieben werden.

```
class Counter {
  private _doors: number
  public static readonly WOOD_FACTORS =
    { 'oak': 80, 'pine': 20}
  constructor({doors = 2}: {doors?: number} = {}) {
    this.doors = doors;
  }
  set doors(newDoorCount: number) {
    if(...) { this._doors = newDoorCount }
    else { throw '...' }
  }
  get doors() { return this._doors) }
}
```

Vererbung und Interfaces. Klassen können in der Konstruktor-Argumentliste angeben, ob ein Parameter ein public/private Property sein soll. Dann spart man den Initialisierungscode.

```
interface IPoint {
  readonly x: number; readonly y: number
}
interface ILikableItem { likes? : number }
class DescribableItem {
  constructor(public description: string) {}
}
class POI extends DescribableItem
  implements IPoint, ILikableItem {
  constructor(public x: number, public y: number,
    description: string, public likes?: number {
    super(description)
  }
}
```

`--strict` enables `noImplicitAny`, `noImplicitThis`, `strictNullChecks` (null and undefined values are not in the domain of every type and are only assignable to themselves) and `strictFunctionTypes` (contravariantly instead of bivariantly).

```
let a: any = true
let nb: number | boolean = "aaa" // fails
const nai = [3, 7, 9]
nai[6] = true // fails
let t: [number, boolean] = [42, false]
enum Decision { yes, no }
let y: Decision = Decision.no
declare let m: number
m = true // fails
function f(n1: number, n2?: number) {} // n2 optional
f(2) // ok
f(2, "ab") // fails
f(2, 3, 7) // fails
```

`tsc --target 'ES6' f.ts` erstellt ein ES6 JS anstatt ein ES3.

9 Testing

9.1 Phasen

Ein Unit Test hat vier Phasen: (1) Setup, (2) Exercise, (3) Verify, (4) Teardown

9.2 Mocha

Test-Beschreibungen geschachtelt.

```
describe('Array', function() {
describe('#indexOf()', function() {
  beforeEach(function() { this.testArray = [1, 2, 3]});});
  it('should return ...', function() {
    expect(this.testArray.indexOf(4)).toBe(-1))
  }); // weitere it()...
}); // weitere describe()...
});
```

9.3 Expect API

```
expect(x). toBe(val). toEqual(val). toThrow(err).
toExist(). toBeTruthy(). toNotExist(). toBeFalsy()
```

9.4 Spies API

```
const video = { play: () => {...} }
spy = expect.spyOn(video, 'play')
expect(spy.calls.length).toEqual(1)
expect(spy.calls[0].context).toBe(video)
```

`spy.expectSpyOn(...). andCallThrough(). andCall(fn). andThrow(exception) .andReturn(value)`
Stub-Werte zurückgeben: `spyOn(Date, 'now').andReturn(currentDate)` (currentDate fixiert).

9.5 Test Pattern

Test Double Pattern DOC (depended-on Component) mit einem Interface ersetzen und einmal real und einmal als Double implementieren.
Test Stub Pattern Immer dasselbe zurückgeben.
Test Spy Pattern

```
class BankAccount { withdraw() {}, deposit() {} }
describe('A new transaction executed', function() {
  beforeEach(function() {
    this.accountA = new BankAccount()
    this.accountB = new BankAccount()
    spyOn(this.AccountA, 'withdraw')
    this.transaction =  new Transaction(
      this.accountA, this.accountB, 25)
  });
  it('withdraws 25 from account A', function() {
    this.transaction.execute()
    expect(this.accountA.withdraw).
      toHaveBeenCalledWith(25)
  });
});
```

Fake Object Pattern Z.B. einen Fake Hash Service bauen, der immer auf dem gleichen Seed operiert. Unterschied zum Stub ist sehr fein.
Mock Object Pattern Objekt, das sich selber verifizieren kann, das wir dann fragen können ob alles korrekt lief

9.6 Empfehlungen

Pure Functions: Testbarkeit optimal, wo nötig Test Doubles als Argumente übergeben. **Non-pure Functions/JS Objekte**: Context für Test fixieren, globale Variablen setzen, Veränderung globaler Variablen überprüfen, Veränderung der Input-Argumente überprüfen, wo nötig Test Doubles als Argumente übergeben.

10 NodeJS/Express Fakten

Module Suchreihenfolge 1. Core-Module, 2. falls mit Slash/Punkt am Anfang: (a. File, b. Directory), 3. falls Filename (`require('mymodule')`): in node_modules, danach bis zum System Root. **node_modules** nicht ausliefern! Kann Binary-Teile beinhalten, besser in package.json.
Middleware Die Reihenfolge der Registrierung bestimmt die Ausführungsreihenfolge. **Static Middleware** es sind auch mehrere static-Routes möglich. **Error-Middleware** muss als letztes registriert werden. Es können mehrere sein, die letzte muss die Anfrage beenden. Wird aufgerufen, wenn Error-Objekt dem Callback übergeben wird. Signatur wie normale Middleware, aber zusätzlicher erster Parameter `error`
Ordnerstruktur vs. Architektur
`/routes` **Entry point** Front Controller (does routing). `/controller` **Controller**. `/services` **Model**.
Token Idee: bei jeder Anfrage ein Token mitgeben. Bildet Berechtigungen ab, hat Ablaufdatum. Bsp: JSON Web Token (JWT).

10.1 ES6 Module

Besitzen eigenen Scope. Ermöglicht Kapselung und lazy-loading. Konventionen: 1 File = 1 Modul (= 1 Unit für Tests); 1 Folder = 1 Package = Sammlung von zusammengehrenden Modulen und Klassen. Modul-Import/Export Syntax ist leicht anders als NodeJS/CommonJS. Module sollten Endung `.mjs` haben. Sie werden im Strict-Mode interpretiert. Nur einmal geladen. Können nur mit Flag `--experimental-modules` in Node genutzt werden, Node unterstützt keine Dynamic Imports.

```
// lib.mjs
export function add(x, y) { return x + y}
// main.js
import {add} from './lib.mjs'; add(2, 3)
```

Dynamic Import `'./main1.mjs'` definiert `global.DEMO_VAR = 'MAIN1'`.

```
import('./main1.mjs').then(() => {
  console.log(global.DEMO_VAR);
});
```

ES6 Module im Browser können mit `<script [src=''] type='module'>[source]</script>` geladen werden.

10.2 Streams

```
let server = http.createServer(function (req, res) {
  let stream = fs.createReadStream(__dirname + '/d.txt');
  stream.pipe(res);
```

10.3 Events

```
const EventEmitter = require('events').EventEmitter;
class Door extends EventEmitter {
  constructor(){ super(); }
  ring() { setTimeout(() => { this.emit('open'); },
    100); };
}
```

10.4 SOP/CORS

Die **S**ame **O**rigin **P**olicy erlaubt XHR-Requests nur zur Origin. Mit **C**ross-**O**rigin **R**esource **S**haring kann man Cross-Site Requests ermöglichen. Chrome erlaubt kein CORS nach Localhost.

11 NodeJS/Express

```
// app.js
var express = require('express')
var bodyParser = require('body-parser')
var session = require('express-session')
var app = express()
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'hbs')
app.use(bodyparser.urlencoded())
app.use(cookieParser())
app.use(session({secret: 'abc', resave: false,
  cookie {secure: true}}))
app.use(express.static(path.join(__dirname, 'public')))
app.use('/', require('./routes/search'))
app.use((req, res, next) => { // catch 404
  const err = new Error('Not found'), err.status = 404;
  next(err) // will skip to error middleware
});
app.use((err, req, res, next) => {
  res.status(err.status | 500)
  res.render('error')
});
module.exports = app
// routes/search.js
const express = require("express")
const router = express.Router()
const controller =
  require("../controllers/searchController")
router.get("/", (req, res, next) =>
  controller.index(req, res))
// ODER router.get("/", controller.index)
router.get("/*", (req, res, next) =>
  controller.index(req, res))
router.get("/search", (req, res, next) =>
  controller.search(req, res))
module.exports = router
// public/js/autocomplete.js
$(() => {
  $("#searchText").keyup(() => {
    const searchText = $("#searchText").val()
    const searchType = $("#searchType").val()
    const resultContainer = $("#resultContainer")

    $.getJSON(
`/search?searchText=${searchText}&searchType=${searchType}`,
      data => {
        if (searchType === "person") {
          // person
        } else {
          resultContainer.empty()
          data.result.forEach(element =>
            resultContainer.append(
              new Option(element.title))
)}})})})
//controllers/searchController.js
const service = require("../services/searchService")
index = (req, res) => {
  res.render("index", { title: "Suche",
    hasResult: false })
}
const isPersonSearch = typeString =>
  (typeString === "person" ? true : false)
const getTitleString = isPersonSearch =>
  isPersonSearch ? "Personensuche" : "Textsuche"
search = (req, res) => {
  const searchText = req.query["searchText"] || ""
  const searchType =
    isPersonSearch(req.query["searchType"])
  const searchResult =
    service.search(searchText, searchType)
  res.format({
    "text/html": () =>
      res.render("index", {
        title: getTitleString(searchType),
        hasResult: true,
        isPersonSearch: searchType,
        searchResult: searchResult
      }),
    "application/json": () =>
      res.send(JSON.stringify(searchResult)),
    default: () => res.status(406).send("Not Acceptable"
  })
}
module.exports = { index, search }

// index.hbs
<script src='js/autocomplete.js'></script>
<form action="/search" method="get">
  <label>
  Suche<input name="searchText" id="searchText"
    list="resultContainer"></label>
  <select name="searchType" id="searchType">
    <option value="text">Text Suche</option>
    <option value="person">Personen Suche</option>
  </select>
  <button>Suchen</button>
</form>
<datalist id="resultContainer" />
<h1>{{title}}</h1>
{{#if isPersonSearch}}
<table class="table">
  <thead>
    <tr><th><!-- Name, Email, Institut --></th></tr>
  </thead>
  <tbody>
    {{#each searchResult.result}}
    <tr>
      <td>{{this.name}}</td>
      <td><a href="mailto:{{this.email}}">
      {{this.email}}</a></td>
      <td>{{this.institut}}</td>
    </tr>
    {{/each}}
    </tbody>
</table>
{{else}} {{#each searchResult.result}}
<h4>{{this.title}}</h4>
<a href="{{this.link}}">{{this.link}}</a>
<p>{{{this.text}}}</p>
{{/each}} {{/if}}
```

Value mit Priorität setzen Hier: 1. Query, 2. Session, 3. Default

```
const getValueFromQuerySessionDefault =
(query, session, defaultValue) =>
  query !== undefined ? query
  : session !== undefined ? session
  : defaultValue
// Anwendung
req.session.bla = getValueFromQuerySessionDefault(
  req.query.bla, req.session.bla, "Default Wert")
```

Datalist für Autovervollständigung

```
<label>Choose a browser from this list:
<input list="browsers" name="myBrowser" /></label>
<datalist id="browsers">
  <option value="Chrome">
  <option value="Firefox">
  <option value="Internet Explorer">
</datalist>
```

Number-Input

```
<input type="number" step="1" min="1" max="99" />
```

12 NeDB

NeDB gibt jedem Doc eine ID unter `_id`. Errors abfangen: `err ? console.log(err.message) : cb(...))`

```
const Datastore = require('nedb')
const db = new Datastore({filename: './d.db',
  autoload: true}) // no need to load
const add = (note, cb) => {
  db.insert(note, (err, newDoc) => cb(newDoc))
}
const getNote = (id, cb) => {
  db.findOne({_id: id}, (err, newDoc) => cb(newDoc))
}
const getAllByDueDateThenImportance = cb => {
  return db.find({finished: true}) // {} for all
  .sort({dueDate: 1, importance: -1})// -1 reversed
  .exec((err, docs) => cb(docs))
}
const updateNote = (id, newNote, cb) => {
  db.update({_id: id}, newNote,
    {returnUpdatedDocs: true},
    (err, count, doc) => cb(doc))
}
const removeNote = (id, cb) => {
  db.remove({_id: id}), // can be any query
  {multi: true}, // delete multiple matching
  (err, count) => cb()
}
```