

# ZUSAMMENFASSUNG

## Lernziele

- Sie kennen die wichtigen Architekturen und Technologien zur Erstellung von regulären Web-Applikationen
- Sie beherrschen grundlegende Techniken für das Design und die Programmierung der Client-Seite von regulären Web-Applikationen
- Sie beherrschen grundlegende Techniken für das Design und die Programmierung der Server-Seite von regulären Web-Applikationen
- Sie besitzen eine Übersicht über grundlegende Frameworks und Tools für die Erstellung von regulären Web-Applikationen
- Sie beherrschen grundlegende Techniken für das Gestalten von mobilen Webseiten
- Sie besitzen grundlegendes Wissen zu Prozessen und Techniken für den User Centered Design von regulären Web-Applikationen

## Unterlagen / Bücher

- E-Bücher, welche das Lernen unterstützen. (Bezüge in den individuellen Vorlesungen)

## Lerninhalte

- **Client / Pure JS**
  - o Responsive Layout mit CSS Flexbox, Media Queries, Viewports(Mobiles)
  - o Wartbares CSS mit CSS Preprocessor (Saas)
  - o Javascript: OO(Prototypes, Constructors); ES6 / Typescript
  - o Code Struktur in JS Module, DRY, JS-Coding Standards
  - o Automatisches Testing
- **Server (Node)**
  - o Session-Handling
  - o Server-Side Rendering / Templating
  - o Form-Handling
  - o Web Security (Node)
  - o 3 Tier, DTO's, Server-based validation
- **Externes Design / UCD**
  - o Fluid & Responsive Layout Design: Patterns und Guidelines
  - o UCD Techniken für Layout & Struktur-Design

## Testat

- Kontinuierlicher Arbeit am Miniprojekt
- Start ab Woche 3
- Details in Woche 2
- Abgabe in Woche 9

## Prüfung

- Closed Book, 1 A4 Zusammenfassung einseitig

<b>EINLEITUNG .....</b>	<b>9</b>
<b>REPETITION WED1 THEMEN .....</b>	<b>9</b>
ENCODING .....	9
HTML SEITENSTRUKTUR.....	9
HTML SPECS.....	9
CSS REGELSPEZIFITÄT .....	10
AJAX.....	10
JAVASCRIPT .....	10
<i>Javascript Scope, Kontext, „this“</i> .....	10
USABILITY.....	11
HANDLEBARS .....	11
<i>Handlebars Template</i> .....	11
<i>Datumskonvertierung</i> .....	11
<i>Handlebars Template</i> .....	11
<i>Script mit index.js</i> .....	11
<i>CSS</i> .....	12
<i>Compare Funktion</i> .....	13
<i>Vertiefung Handlebars</i> .....	13
<b>NODEJS.....</b>	<b>14</b>
DER KЛАSSISCHE WEBSERVER.....	14
WAS IST NODEJS .....	14
EVENT-DRIVEN, NON-BLOCKING.....	14
GRUNDLAGEN .....	14
<i>Asynchrone Programmierung / Callback</i> .....	14
<i>Event</i> .....	14
<i>Core Modules</i> .....	14
ERSTES BEISPIEL .....	14
<i>Module in Node.js</i> .....	15
<i>Export</i> .....	15
<i>Module</i> .....	15
<i>Zyklus</i> .....	16
API .....	16
<i>Streams</i> .....	16
<i>Events</i> .....	16
FAVICON.....	16
<b>EXPRESS JS / AJAX .....</b>	<b>18</b>
PATTERNS.....	18
<i>MVC-Pattern</i> .....	18
<i>Front Controller</i> .....	18
EXPRESS.....	18
<i>Installation</i> .....	18
<i>Middleware</i> .....	18
<i>Middleware registrieren</i> .....	19
<i>Die Middleware „Conntect“</i> .....	19
DEMO 1 – ROUTING.....	19
<i>Router-Middleware</i> .....	19
<i>Static-Middleware</i> .....	19
<i>Custom-Middleware</i> .....	20

Error-Middleware .....	20
DEMO 2/3 – MODEL .....	20
Struktur .....	20
Model .....	20
NoSQL-Datenbanken am Beispiel nedb .....	21
ODM (object document mapping) mit Mongoose .....	21
DEMO 4 – VIEW .....	22
Template Engine .....	22
DEMO 5 – SESSION & SECURITY .....	22
Cookie .....	22
Session .....	22
Cookie-Tools .....	22
Session .....	22
DEMO 6 – REST .....	23
Unterschiedliche Formate .....	23
Token .....	23
DEMO 6 – AJAX .....	24
Motivation .....	24
Definitionen .....	24
Nachteile .....	24
XMLHttpRequest(XHR) .....	24
JQuery.ajax .....	24
SOP / CORS .....	25
WEBSOCKETS .....	25
GENERATOR .....	25
DEPLOY .....	25
<b>REST .....</b>	<b>26</b>
WAS IST REST .....	26
Richardson's Maturity Model .....	26
Die Welt vor REST .....	26
Eigenschaften von REST .....	26
ROA: RESSOURCE ORIENTED ARCHITECTURE .....	28
Grundprinzipien .....	28
Ressource Name .....	28
Ressource Links .....	28
Benutze Standard-Methoden .....	29
Ressource Repräsentation .....	29
HATEOAS (HYPERMEDIA AS THE ENGINE OF APPLICATION STATE) .....	29
VERSIONIERUNG .....	30
BEST PRACTICES .....	30
<b>FLEXIBLES LAYOUT MIT CSS &amp; FLEXBOX .....</b>	<b>31</b>
MOTIVATION .....	31
FLEXIBLES VS. RESPONSIVES LAYOUT .....	31
Layout .....	31
Flexibles Layout .....	31
Responsives Layout .....	31
FLEXIBLES LAYOUT – CSS GRUNDLAGEN .....	32
CSS Box Model .....	32
Scrolling beim CSS Box Model .....	32
Werte von Positionen .....	32

<i>Flexibles Layout mit absoluter Positionierung</i> .....	32
<i>Eigenschaften von Block / Inline / Inline-Block Elementen</i> .....	33
<b>AKTUELL WICHTIGSTE CSS LAYOUT TECHNIK</b> .....	33
<i>Flexbox – Display</i> .....	33
<i>Flexbox – Size</i> .....	34
<i>Flexbox – Direction</i> .....	34
<i>Flexbox – Wrap</i> .....	34
<i>Flexbox – Order</i> .....	35
<i>Flexbox – Direction</i> .....	35
<i>Browser Support</i> .....	35
<i>Einschränkung von Layouting mit Flexbox</i> .....	35
<b>ANALYSE REALISTISCHER LAYOUTS</b> .....	36
<b>WEITERE TECHNIK ZUR POSITIONIERUNG / LAYOUT: FLOATS (VERGANGENHEIT)</b> .....	37
<i>Details (kein Lernstoff)</i> .....	37
<b>WEITERE TECHNIK ZUR POSITIONIERUNG / LAYOUT: DISPLAY TABLE / TABLE-CELL</b> .....	37
<b>WEITERE TECHNIK ZUR POSITIONIERUNG / LAYOUT: CSS GRID (AUSBlick)</b> .....	37
<b>WEITERE PRAXISTIPS: ZENTRIEREN VON ELEMENTEN (HORIZONTAL &amp; VERTIKAL)</b> .....	37
<b>LAYOUT TESTING</b> .....	38
<i>Chrome Debug Tools</i> .....	38
<i>Emulatoren und Simulatoren</i> .....	38
<i>Hardware</i> .....	38
<b>BROWSER SUPPORT: PROGRESSIVE ENHANCEMENT &amp; GRACEFUL DEGRADATION</b> .....	38
<i>Graceful Degradation</i> .....	38
<i>Progressive Enhancement</i> .....	39
<i>Mobile First</i> .....	39
<b>RESPONSIVE LAYOUT MIT CSS MEDIA-QUERIES</b> .....	40
<b>RESPONSIVE LAYOUT MIT MEDIA QUERIES</b> .....	40
<i>Einheiten</i> .....	40
<i>Media-Attribut in link Referenzen auf externe Stylesheets</i> .....	41
<i>Mobile First Layout mit Media-Queries</i> .....	41
<i>Prozess der Erstellung von Media Queries</i> .....	41
<i>CSS Pixel</i> .....	42
<i>Abfrage von spezifischen Geräten</i> .....	42
<i>Weitere «Tipps &amp; Tricks» zu Layout &amp; Media Queries</i> .....	42
<b>VIEWPORT</b> .....	43
<b>RESPONSIVE IMAGES</b> .....	43
<i>Code mit img</i> .....	43
<i>Code mit picture</i> .....	44
<i>Support</i> .....	44
<i>Polyfill</i> .....	44
<i>Einschub – Performance Tool</i> .....	44
<b>SASS</b> .....	45
<b>CSS PRÄPROZESSOR</b> .....	45
<i>Vorteile / Möglichkeiten</i> .....	45
<b>INSTALLATION</b> .....	45
<b>SASS vs. SCSS</b> .....	45
<b>FEATURES</b> .....	46
<i>Spezielle Zeichen – Übersicht</i> .....	46
<i>Variablen</i> .....	46
<i>Nesting</i> .....	47

<i>Parent-Selektor</i>	47
<i>Partials / Import</i>	48
<i>Mixings</i>	48
<i>Extend / Inheritance</i>	49
<i>Programmierung</i>	50
<b>RESPONSIVE WEB-DESIGN .....</b>	<b>51</b>
KURZREPETITION WED1 USABILITY + USER CENTERED DESIGN .....	51
HERAUSFORDERUNGEN FÜR RESPONSIVE WEB-SITES GEORDNET NACH GARRET EBENEN.....	51
<i>Analyse von Beispielen</i> .....	52
RESPONSIVE LAYOUT PATTERNS .....	53
<i>Reflow nach dem Prinzip «Mostly Fluid»</i> .....	53
<i>Reflow nach dem Prinzip «Column Drop»</i> .....	54
<i>Reflow nach dem Prinzip «Layout Shifter»</i> .....	54
«Reflow» Eine Spalte (Mobil) und viele Spalten (Desktop) .....	54
«Expand» Maximale Breite + Rand (Tablet/Desktop).....	55
Sidebar/SidePic für Landscape Ansicht .....	55
Master dann Detail(Mobile) und Master inklusive Detail (Tablet/Desktop) .....	55
<i>Reflow nach dem Prinzip «Off Canvas»</i> .....	56
Off-Screen Menu(Mobile) und On-Screen Menu (Tablet/Desktop).....	56
<i>Layoutprobleme</i> .....	56
RESPONSIVE MICRO-PATTERNS .....	57
<i>Progressive Disclosure</i> .....	57
<i>Responsive Tables</i> .....	57
<i>FixText</i> .....	58
<i>Mehr Patterns (nicht nur Responsive)</i> .....	58
(RESPONSIVE) FORM DESIGN .....	58
<i>Einfach mit Flexbox</i> .....	58
<i>Platz sparen mit "Float Labels»</i> .....	59
«Review Mode» ohne Inputfelder spart Platz.....	59
<i>Web Formular Design Generell</i> .....	59
<i>Ratschläge zu guten Fehlermeldungen</i> .....	60
<i>Bad Practice</i> .....	60
MATERIAL DESIGN.....	61
<b>UCD VERTIEFUNG.....</b>	<b>62</b>
SENSORISCHES DESIGN – ACCESSIBILITY.....	62
<i>Beispiele</i> .....	64
<i>Accessibility Regeln – Technologieunabhängig</i> .....	64
<i>Wichtigste Regeln um HTML Accessibility zu verbessern</i> .....	64
INTERAKTIONSDESIGN – DESIGN VON CONTROLS.....	65
<i>Affordances von Controls – Was wird verstanden</i> .....	65
<i>Beispiele von Affordance</i> .....	66
STRUKTUR DESIGN – CARD SORT ETC.....	66
<i>Navigationsstruktur mit Cardsort bestimmen am Beispiel des HSR Intranet mit OptimalSort</i> .....	67
<i>Alternative bei bestehenden Sites – Tree Testing</i> .....	67
UMFRAGE UND STRATEGIE.....	68
<i>Scenarios &amp; Wire-framing / Testing</i> .....	68
<i>Fragebögen</i> .....	70
<b>WEB SECURITY GRUNDLAGEN .....</b>	<b>71</b>
INTRO .....	71

Web Engineering + Design 2	
<i>Mini Fallstudie – Yahoo XSS Vulnerability</i>	71
<i>Web Security = OWASP Top 10</i>	71
CROSS-SITE-SCRIPTING (XSS)	71
<i>Definition</i>	71
<i>Demobispiel mit Node (Node-Goat)</i>	71
<i>Übersicht Kontext-sensitives Encoding gegen XSS</i>	72
<i>Weitere Sicherungsmassnahmen gegen XSS</i>	72
JS CODE INJECTION	73
<i>Mini Fallstudie – JS Injection beim BitTorrent Client Popcorn Time</i>	73
<i>Definition</i>	73
<i>Demo von NodeGoat</i>	73
<i>Weitere Massnahmen</i>	74
BROKEN AUTHENTICATION AND SESSION MANAGEMENT	74
<i>Definition</i>	74
<i>Beispiel 1</i>	74
<i>Beispiel 2</i>	74
<i>Beispiel 3</i>	74
INSECURE DIRECT OBJECT REFERENCES	75
<i>Definition</i>	75
<i>Beispiel mit NodeGoat</i>	75
CROSS SITE REQUEST FORGERY	75
<i>Definiton / Therorie</i>	75
<i>Beispiel an NodeGoat</i>	76
TOP OVERLOOKED SECURITY THREATS TO NODE_JS WEB APPLICATIONS PRESENTATION	76
<b>LAYERING IM BROWSER</b>	<b>77</b>
MVC BEISPIEL EINES COUNTERS	77
MVC BEISPIEL EINES MULTI-COUNTERS (DYNAMIC VIEW & MODEL CREATION)	77
<b>OO PROGRAMMIERUNG MIT JAVASCRIPT</b>	<b>78</b>
WARUM?	78
PROPERTIES, METHODEN UND «THIS»	78
<i>Was ist "this"</i>	79
<i>Achtung beim "Abtrennen" von Methoden, welche «this» nutzen</i>	79
<i>Achtung in Methoden mit anonymen Funktionen</i>	79
VERERBUNG & TYPENERWEITERUNG	80
<i>Primitive Types</i>	80
<i>Reference Types</i>	80
<i>Konstruktor-Funktionen definieren neue Typen</i>	81
<i>Übersicht – Methoden für die Erstellung von JS Objekten</i>	81
<i>Prototypen – Objekte für die effiziente Speicherung von Methoden</i>	81
<i>Private Eigenschaften</i>	81
<i>Beispiel</i>	82
<i>Mehrstufige Vererbung und die Prototype Chain</i>	82
<b>ES6 / ECMASCIPT 2015, FUNCTIONALS JS, JS CODING STYLEGUIDES</b>	<b>83</b>
ECMASCIPT 2015 (ES6)	83
<i>Grundlagen</i>	83
<i>OO mit ES6</i>	87
FUNKTIONALES JAVASCRIPT	87
<i>Theorie</i>	87
<i>Beispiele</i>	88

JS STYLEGUIDES .....	88
<b>JS MODULE .....</b>	<b>89</b>
MODULE THEORIE .....	89
<i>Definition</i> .....	89
<i>Vorteile von Modulen</i> .....	89
<i>Abgrenzung des Begriffs</i> .....	90
<i>Typische Konventionen bei JS-Modulen</i> .....	90
<i>Tiers vs. Layers vs. Classes vs. Services vs Modules / Packages</i> .....	90
JS-MODULE: NUTZUNG AUF DEM NODE-SERVER (COMMONJS).....	90
JS-MODULE: ANSÄTZE FÜR MODULE IM BROWSER (COMMONJS).....	90
<i>Revealing Module Pattern</i> .....	90
<i>Nutzung von Require-JS</i> .....	91
<i>Nutzung ES6 Import/Export</i> .....	93
<b>TYPESCRIPT .....</b>	<b>94</b>
MOTIVATION UND GRUNDLAGEN .....	94
DETAILS – TYPEN UND SYNTAX-MODULE.....	94
<i>Variablendeklarationen mit Basistypen</i> .....	94
<i>Variablendeklarationen mit komplexen Typen</i> .....	95
<i>Funktionsdeklarationen</i> .....	95
<i>Komplexe Typen – Klassen</i> .....	95
<i>Klassen und Interfaces</i> .....	96
<i>Module in Typescript</i> .....	97
ABSCHLUSS .....	97
<b>TESTING.....</b>	<b>98</b>
UNIT TEST PATTERNS .....	98
<i>Test Double Pattern</i> .....	98
<i>Test Stub Pattern</i> .....	98
<i>Test Spy Pattern</i> .....	99
<i>Fake Object Pattern</i> .....	99
<i>Mock Object Pattern</i> .....	99
TESTEN MIT JASMINE.....	100
<i>Jasmine und BDD</i> .....	100
<i>BDD User Stories</i> .....	100
<i>BDD Story versus Specification</i> .....	100
<i>Jasmine API</i> .....	100
<i>Mehr zu Jasmine API</i> .....	101
TEST-DRIVEN DEVELOPMENT .....	101
<i>Einführung</i> .....	101
<i>Der Zyklus</i> .....	101
TEST-AUTOMATION.....	101
<i>Continuous Integration</i> .....	101
<i>Continuous Delivery/Deployment</i> .....	102
<i>Mögliche eingesetzte Tools</i> .....	102
<i>Tool Chain für Angular 2</i> .....	102



## Einleitung

### Architektur von Web-Anwendungen

Front-End	Mobile Apps, Webseite, Desktop GUI
Network	WLAN, Firewalls, Router, VPN, CDN
Back-End	Server (Web, App), Datenbanken, Integration

WED1 Wissen ist Voraussetzung für WED2 (jQuery, JS, HTML und CSS, Ajax, Event Handling, JSON, http REST sowie Externes Design / UCD)

<b>WED2 Themen</b>	<b>Client / Pure JS</b> (Responsive Layout, Wartbares CSS mit Preprocessor, Javascript OO, Code Struktur in JS Module, Automatisches Testing), <b>Server / Node</b> (Session-Handling, Server-Side-Rendering/Templating, Form-Handling, Web-Security, 3 Tier) und <b>Externes Design</b> (Fluid & Responsive Layout Design: Patterns und Guidelines, UCD Techniken für Layout und Struktur-Design)
--------------------	--

**Prüfung** von 120 Minuten, Testat als Zulassungsbedingung

### Repetition WED1 Themen

Dies sind Aufgaben aus der Prüfung von Modul WED1 aus der Durchführung FS 2016. Genauere Details sind den Vorlesungsunterlagen vom Modul WED1 zu entnehmen.

#### Encoding

Aus einer bestimmten Bytefolge lässt sich nicht schliessen, welches Encoding verwendet wurde. UTF-8 ist «variable-length». Dies bedeutet das die Zeichen in unterschiedlichen Längen abgelegt werden. UTF-8 verbraucht für normale Zeichen (ASCII) gleich viel Speicher. Wenn immer möglich ist UTF-8 zu verwenden.

#### HTML Seitenstruktur

Es sollte grundsätzlich auf header, nav, main, aside und footer auf oberster Ebene verteilt werden. Für allfällige Inhalte sind die spezifischsten Tags (address, ... ) zu verwenden. Innerhalb von einem Artikel ist eine weitere Verteilung auf header und footer in den meisten Fällen sinnvoll. (Vor allem in den Fällen für eine Artikelseite oder Newsseite).

#### HTML Specs

Grundsätzlich lassen sich mit dem HTML Content Model und den Specs bestimmen ob Tags erlaubt sind und allfällige Teile weggelassen werden dürfen. So darf beispielsweise der End-P Tag weggelassen werden, falls direkt ein Blockquote folgt.

Examples:

```

*           /* a=0 b=0 c=0 -> specificity =  0 */
LI          /* a=0 b=0 c=1 -> specificity =  1 */
UL LI       /* a=0 b=0 c=2 -> specificity =  2 */
UL OL+LI   /* a=0 b=0 c=3 -> specificity =  3 */
H1 + *[REL=up] /* a=0 b=1 c=1 -> specificity = 11 */
UL OL LI.red /* a=0 b=1 c=3 -> specificity = 13 */
LI.red.level /* a=0 b=2 c=1 -> specificity = 21 */
#x34y      /* a=1 b=0 c=0 -> specificity = 100 */
#s12:not(FOO) /* a=1 b=0 c=1 -> specificity = 101 */

```

```

</head>
<body>    f:8px; c:blue
    <span id="first">Text 1</span>
    <div id="all">f:2em -> 16px
        <div>
            <span class="between1">Text 2</span>
            <span class="between2">Text 3</span>
        </div>
        <div id="second-last">Text 4</div>
        <p class="text">Text 5</p>
    </div>
</body>
</html>

```

```

<!DOCTYPE html>
<html>
<head lang="en">
<meta charset="utf-8" />
<style>

body (           /* Regel A */
    font-size: 8px;
    color : blue;
)

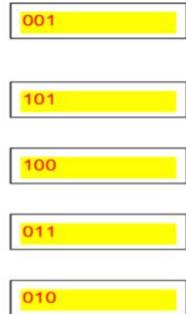
#all > div (   /* Regel B */
    font-size : 2em;
)

#all * (         /* Regel C */
    font-size : 0.3em;
)

span:not(.between1) /* Regel D */
    color : red;
)

.text {
    color: green;
    font-size : 0.5rem !important;
}
</style>

```



	CSS Regeln, die dieses Element DIREKT betreffen (z.B. A,D,E) (2.5P)	Regeln, die dieses Element NUR indirekt beeinflussen (Vererbung) (2.5P)	Font-Size in Pixel (3.75P)	Farbe (3.75P)
Text 1				
Text 2				
Text 3				
Text 4				
Text 5				

Text 1	D	A	8px	red
Text 2	C	A, B	4.8px	blue
Text 3	D, C	A, B	4.8px	red
Text 4	B, C	A	16px	blue
Text 5	C, E	A	8px	green

## AJAX

Für AJAX gibt es verschiedene Shorthands. `$.post()` ist ein Shorthand für `$.ajax({type:'POST',....})`. AJAX Request sollten immer asynchron benutzt werden. Mittels AJAX können beliebige Dateiformate transportiert werden. Bei AJAX gilt grundsätzlich die Same Origin Policy. Nicht gilt: andere Portangaben, IP anstatt Domain, https oder eine andere Domain.

## Javascript

**Hoisting**, Bei Hoisting werden die Deklarationen von Methoden / Variablen an den Anfang des Scopes verschoben. Die Initialisierung werden dabei aber nicht ge-„hoisted“. Dies kann zu Bugs führen. Dabei ist zu beachten das auch Funktionsdefinitionen verschoben werden.

### Javascript Scope, Kontext, „this“

```

var car = {
    numberOfWorks: 4,
    getNumberOfWheels: function () {
        return this.numberOfWorks;
    }
};
var bike = {
    numberOfWorks: 2
};

console.log("Ausgabe A:", car.getNumberOfWheels(), bike.numberOfWorks);

var wheelsGetter = car.getNumberOfWheels;
console.log("Ausgabe B:", wheelsGetter());

console.log("Ausgabe C:", wheelsGetter.apply(bike));

```

Ausgabe A: 4 2  
Ausgabe B: undefined  
Ausgabe C: 2

Problem	Stone (Nr)	Garrett (Nr)
Die Suche auf der HSR Homepage funktioniert nicht gleich wie die Suche im HSR Intranet.	■ 6, 3, 7	3, 4
Bei der Suche im Intranet wird auch bei der Selektion des Radio-Buttons „Website“ hinter dem Suchfeld stets das Tab mit den gefundenen Personen zuerst angezeigt.	■ 3, 1, 2, 6	2, 4, (1)
Bei der Suche werden auch veraltete Seiten gefunden die in der Navigationshierarchie nicht mehr verlinkt sind.	■ 4, 3	4, 3
Beim Eintippen von „semester“ wird ein Quick-Link „semesterplan“ angezeigt, welcher zur Seite „Semesterreckdaten“ weiterleitet. Wird dagegen nach „semesterplan“ gesucht, werden keine Einträge gefunden.	■ 7, 6, 5	3, 4
Das „pdf“ Icon vor Dokumenten ist so schlecht sichtbar, dass nicht klar wird, dass es sich um einen Download-Link handelt.	■ 2, 1, 6	1
Die Informationen für Erstsemestrigäte sind zu finden unter HSR-intern > IT-Informatikdienste > Support / IT-ServiceDesk > Einstieg für Erstsemestrigäte.	■ 5	3, 5
Die Intranet-Startseite hat keinen speziellen Eintrag für Erstsemestrigäte.	■ 1, 5	5, 4, 1

Stone Kriterien	Garrett Ebenen
S1) Visibility	G1) Oberfläche
S2) Affordance	G2) Raster
S3) Feedback	G3) Struktur
S4) Simplicity	G4) Umfang
S5) Structure	G5) Strategie
S6) Consistency	
S7) Tolerance	

## Handlebars

Für die Handlebars muss grundsätzlich das zugehörige JS eingebunden werden. Ebenfalls ist jQuery nötig.

### Handlebars Template

#### Datumskonvertierung

```
//usage: {{dateFormat someDate 'YYYY MM DD'}}}
Handlebars.registerHelper('dateFormat', function(date, format) {
  if (window.moment) {
    return moment(date).format(format);
  } else{
    return date.toString();
  }
});
```

### Handlebars Template

```
<script id="table-template" type="text/x-handlebars-template">





```

CSS

```
.quote.positive::before{
  content: '▲';
  display: inline-block;
}

.quote.negative::before{
  content: '▼';
  display: inline-block;
}

.quote.positive{
  text-decoration: overline;
}

.quote.negative{
  text-decoration: underline;
}
```

### Script mit index.js

```
$
(function() {
  var orderDesc = true; //current sort order

  function getAktienKurseFromServer( numberOfMonths = 3 , callback ) {
```

---

\$.get("/api/aktienkurse", {monate: numberOfMonths}, callback, "json");

---

... oder \$.get("/api/aktienkurse?monate="+numberOfMonths, callback, "json");

---

}

```

(function($){
    var orderDesc = true; //current sort order

    function getAktienKurseFromServer(numberOfMonths = 3, callback) {
        $(function() {
            var container = $("#container");

            var selectedNumber = $("#number-select");
            var template = Handlebars.compile($("#table-template").html());

            function renderData(data) {
                data.aktienkurse.sort((a,b) => orderDesc ? compare(a.firma, b.firma) : compare(b.firma, a.firma));
                container.html(template(data));
            }
        });
    }

    function getDataAndRenderTable() {
        getAktienKurseFromServer(selectedNumber.val(), renderData);
    }

    function getAktienKurseFromServer(numberOfMonths = 3, callback) {
        [...]
        selectedNumber.on("change", getDataAndRenderTable);
        container.on("click", ".title-header", function(){
            orderDesc = !orderDesc;
            getDataAndRenderTable();
        });
        getDataAndRenderTable();
    }
})(jQuery);

```

## CSS

## 6.4 Vervollständigen Sie: styles/index.css

Schauen Sie sich die CSS Anhänge an.

```

*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
/* TODO CSS */
body{
    font-family: arial, sans-serif;
}
table{
    border-collapse: collapse;
    width: 100%;
}
th, td{
    padding: 5px;
    text-align: left;
}
tbody tr:nth-child(2n){
    background: lightblue;
}
thead tr{
    font-weight: bold;
}
.quote.positive::before{...}
.quote.negative::before{...}
.quote.positive{ ... }
.quote.negative{ ... }

```

Aktienkurs-Portal			
Anzahl Monate	2016 Jan	2016 Feb	2016 Mar
Firma			
Microsoft Corp.	▼ 35	▲ 49	▲ 49
Amazoncom	▲ 988.4	▲ 1482.6	▲ 1630.86
Alphabet Inc. (C)	▲ 1335	▼ 934.5	▲ 1308.3

```

main{
    margin-bottom: 2rem;
    /*Bonus - damit der Footer nie
    /*das Main überdecken kann */
}
form{
    margin: 10px 0 10px 0;
}
footer{
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0; /*Alternative: width: 100%*/
    height: 1.9rem;
    line-height: 1.9rem;
}

```

```
function compare(a, b) {  
    if (a > b) {  
        return -1;  
    }  
    if (a < b) {  
        return 1;  
    }  
    return 0;  
}
```

Vertiefung Handlebars

### Was ist besser

Grundsätzlich sollten mit dem Template nur Teil der Seite ersetzt werden (z.B. Tabelle mit den Daten).

### Was braucht es bis etwas in der Seite erscheint?

- Schritt 0:* Template-Text erstellen (Variablen)
- Schritt 1:* Template-Text aus Script Tag holen
- Schritt 2:* CreateHTML Funktion generieren
- Schritt 3:* CreateHTML Funktion mit Daten aufrufen und HTML erhalten
- Schritt 4:* HTML auf Seite platzieren

## Der klassische Webserver

- Ausliefern von statischen Files
- Dynamische Erstellen von HTML Seiten via Server-Side Scripting
  - o JSP, PHP, ASP
  - o Problem: Unterschiedliche Programmiersprache auf Client und Server

Das Problem besteht darin, dass es immer höhere Anforderungen an das Front-End gibt.

## Was ist NodeJS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

## Event-Driven, Non-Blocking

Führer gab es nur einen Prozess / Thread, welche Anfragen bearbeitet haben. Heute gibt es mehrere Threads, jene Anfragen bearbeiten.

In NodeJS wird nur der Transfer ausgeführt. Das heisst er liefert die Anfrage ab und wird benachrichtigt, wenn die Aufgabe fertig ist. Somit kann er während der Wartezeit andere bearbeiten.

Methoden unterscheiden sich durch Kosten (Memory), Grösse der Aufgaben und Skalierung.

## Grundlagen

## Asynchrone Programmierung / Callback

```
function myFunc(a, b, fn) {
  setTimeout(function() {
    fn(a + b);
  }, 1000);
}

myFunc(2, 4, console.log);
myFunc(10, 3, console.error);
```

Funktionen können als Parameter übergeben werden. Diese Funktion wird später aufgerufen. Callback-Hell: Eine Lösung dafür wären es Promises. Die Funktion gibt ein Versprechen, dass er wieder aufgerufen wird. (Code siehe Ablage).

## Event

Ein Event ist ein Pattern. Callbacks sind 1 zu 1 Verbindungen, während Events 1 zu \* Verbindungen sind.

```
$(button).on('click', function(event) {
  console.log("1. subscription");
});

$(button).on('click', function(event) {
  console.log("2. subscription");
});

$(button).on('click', function(event) {
  console.log("3. subscription");
});
```

## Core Modules

Dazu zählen HTTP/HTTPS, URL, FS:FileSystem, Console, UDP/Net, Crypto und DNS. Ausführliche Dokumentationen sind unter <https://nodejs.org/api> verfügbar.

## Erstes Beispiel

<b>Verfügbares auf dem Request (Incoming)</b>	<b>Verfügbares auf der Response</b>
---	-------------------------------------

- **method**
  - GET, PUT, POST, ...
- **url**
  - Angefragte url z.B. /help or help?page=1

- **writeHead**
  - response.writeHead(200, {'Content-Length': body.length,'Content-Type': 'text/plain'});
- **setHeader**
  - response.setHeader("Content-Type", "text/html");
- **statusCode**
  - response.statusCode = 404;
- **statusMessage**
  - response.statusMessage = 'Not found';
- **write**
  - response.write("Data")
- **end**
  - response.end("Data")

## Module in Node.js

Node verwendet commonjs.org. Aktuelle wird kein ES6 Syntax für die Module unterstützt. Für die Modul-Verwaltung wird npm verwendet. Vor dem Beginn muss daher npm init bzw. npm install durchlaufen werden.

Die Module-Files werden nur einmal geladen.

### Export

#### Export:

```
module.exports = myExportedObject;
```

#### Beispiel:

```
var counter = 0;
function add(){ return ++counter; }
function get(){ return counter; }

module.exports = { count : add, get : get };
```

Das ganze wird nur einmal durchlaufen. Das Objekt wird „gecached“ und bei jeder Nachfrage wieder zurückgeben. Das Objekt wird sozusagen geshared und kann von anderen Module theoretisch angepasst werden.

### Module

#### Import:

```
var myModule = require('moduleName');
var myModule = require('./moduleName.js');
```

## Import Reihenfolge

1. Core Module z.B. require(«HTTP»)
2. Falls der mit mit «.» «..» oder «» startet z.B. require(«.myModule»)
  1. Suche als File
  2. Suche als Directory
3. Falls ein «Filename» angeben wurde z.B. require(«myModule»)
  1. Wird ein Module im «node\_modules» gesucht
  1. Wird gesucht bis zum «File-System-Root»

**Merke:** Zyklus sollten verhindert werden.

```
main.js:  
var a = require("./a");  
var b = require("./b");  
a.showObject();  
b.showObject()  
  
a.js:  
var b = require("./b");  
console.log("A");  
module.exports = {showObject : function(){ console.log("A", b)} };  
  
b.js:  
var a = require("./a");  
console.log("B");  
module.exports = {showObject : function(){ console.log("B", a)} };
```

## API

Node hat eine sehr umfangreiche API. Ein typisches Beispiel ist folgendes:

Der Callback ist immer das letzte Argument. Das erste Argument ist, im Fehlerfall, der Error. Der Callback wirft keine „Exception“.

Fast alle async-Methoden haben auch eine Synchrone Variante. z.B. readFileSync. Diese wirst aber eine Exception bei einem Fehler.

## Streams

Das Request und das Response Objekt basieren auf dem Stream. Streams sind „chainable“. Es ist ein gemeinsames Interface für alle „Streaming“ Dienste vorhanden (Netzwerk und File).

Dabei gibt es unterschiedliche Varianten von Streams: readable, writable, transform, Duplex und „classic“.

```
/*Ohne Streams*/  
var server = http.createServer(function (req, res) {  
  fs.readFile(__dirname + '/data.txt', function (err, data) {  
    res.end(data);  
  });  
});  
  
/*Mit Streams*/  
var server = http.createServer(function (req, res) {  
  var stream = fs.createReadStream(__dirname + '/data.txt');  
  stream.pipe(res);  
});
```

## Events

Node hat ein Modul „event“, welches ein „EventEmitter“ beinhaltet. Klassen können vom EventEmitter erben um dessen Eigenschaften zu übernehmen.

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

## Favicon

Steht kurz für „favorite icon“. Die Grösse ist in der Regel quadratisch. Das Format ist entweder ico, png, jpg oder gif.

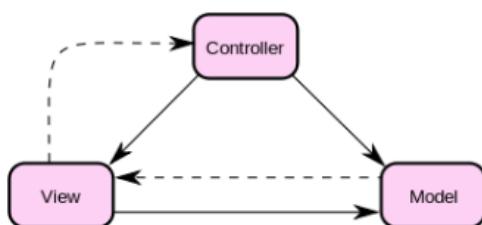
```
<link rel="shortcut icon" type="image/gif" href="facepalm.gif" />
```

Oder platziert das File auf dem Server am richtigen Ort.



## Patterns

## MVC-Pattern

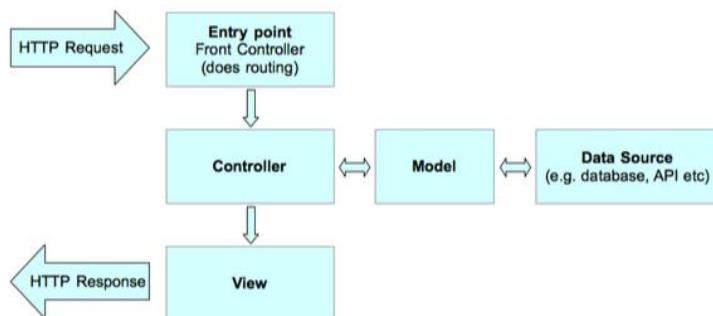


Model: Daten und Datenaufbereitung

Controller: Verknüpft die View mit den Daten

View: Darstellen der Daten

## Front Controller



## Express

## Installation

Dies geht über npm install express. Eine spezifische Version kann wie folgt installiert werden: npm install [express@3.3.8](#).

Genutzt werden kann es wie folgt

```

var http = require('http');
var express = require('express');
var app = express();

http.createServer(app).listen(3000);
  
```

## Middleware

Express nutzt Middleware für die Request Bearbeitung. Er nutzt die Middleware „Connect“. Connect fügt Node das Feature „Middleware“ hinzu. Middleware ist ein Stack von Anweisungen, welche für ein Request ausgeführt wird.



Mit app.use(...) wird eine neue Middleware registriert. Die Reihenfolge der Registrierung bestimmt die Ausführungsreihenfolge.

```
var express = require('express');
var bodyParser = require('body-parser');

var app = express();
var router = express.Router();

app.use(express.static(__dirname + '/public'));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(router);
```

Die Middleware „Connect“

Gibt folgende Inhalte body-parser, cookie-parser, cookie-session, errorhandler, method-override oder serve-static.

Die Middleware „Router“ erweitert den Request mit params, id und get sowie die Response mit sendFile, cookie, format und json.

### Demo 1 – Routing

#### Router-Middleware

Die Middleware befindet sich auf dem Express Objekt.

```
var express = require('express');
var router = express.Router;
```

Folgende Methoden sind wichtig:

#### router.all(path, [callback, ...] callback)

- Wird unabhängig vom der HTTP Methode aufgerufen

#### router.METHOD(path, [callback, ...] callback) (METHOD : get, put, post, delete)

- Wird aufgerufen, falls die jeweilige HTTP Methode verwendet wurde

```
router.get('/', function(req, res){
  res.send('Hello world');
});
```

#### router.route(path)

- Kann benutzt werden um für einen Path verschiedene Methoden zu gruppieren

```
app.route('/book')
  .get(function(req, res) {res.send('Get a random book');})
  .post(function(req, res) {res.send('Add a book');})
```

#### router.all(path, [callback, ...] callback)

- Verwendet Pattern matching. Beispiele:

- '/ab\*cd'
- '/'

#### ■ Dynamische Werte

- '/orders/:id/'
- Der Wert, welcher in :id geparsed wurde, wird in req.params.id abgespeichert

- Es können mehrere Callbacks als Chain übergeben werden.

All diese Methoden liegen auch direkt auf der Express-Applikation.

#### Static-Middleware

Die Aufgabe ist es statische Files auszuliefern. Sie wird wie folgt genutzt:

```
app.use(express.static(__dirname + '/public'))
```

Es sind auch mehrere static-routes möglich.

Sie hat 3 Parameter. Request, response sowie next. Next zeigt auf die nächste Middleware im Stack.

Next kann aufgerufen werden, um die nächste Middleware aufzurufen. Dies kann auch unterlassen werden. In diesem Falle sollte die Anfrage beantwortet werden.

```
function myDummyLogger( options ){
  options = options ? options : {};
  return function myInnerDummyLogger(req, res, next)
  {
    console.log(req.method + ":" + req.url);
    next();
  }
}
```

### Error-Middleware

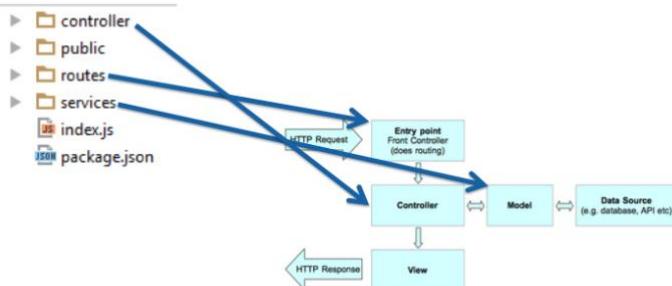
Es ist die Aufgabe, Errors zu bearbeiten, welche von den Middlewares generiert werden. Details dazu können der Online-Dokumentation entnommen werden.

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Die Error-Middleware muss 4 Parameter haben. Error, request, response und next. Die Error-Middleware sollte als die letzte Middleware registriert werden. Es können auch mehrere Error-Middlewares registriert werden. Sie wird aufgerufen, falls ein Error-Objekt dem Next-Callback übergeben wird. Ab diesem Zeitpunkt werden keine normalen Middlewares mehr aufgerufen.

### Demo 2/3 – Model

#### Struktur



#### Model

Ziel ist es, dass die Daten in einem Modul verwaltet und abgespeichert werden. Dazu gibt es verschiedene Möglichkeiten die Daten zu speichern.

- In Memory → Array
- JSON
- NoSQL-Datenbanken
- SQL-Datenbanken

NoSQL-Datenbanken sind Dokumenten-basierend. Jedes Dokument beinhaltet alle Daten, welche notwendig sind. Die Relationen können über „keys“ manuell erstellt werden. Die Relationale-Integrität muss die Applikation selbst sicher stellen oder zumindest damit umgehen können!

### Beispiel „Laden der Datenbank“

```
var Datastore = require('nedb');
var db = new Datastore({ filename: './data/order.db', autoload: true });
```

### Einfügen

Beim Einfügen wird ein Feld `_id` gesetzt, welches die eindeutige ID von der Datenbank beinhaltet.

```
db.insert(order, function(err, newDoc) {
  if(callback){
    callback(err, newDoc);
  }
});
```

### Suchen

find oder findAll

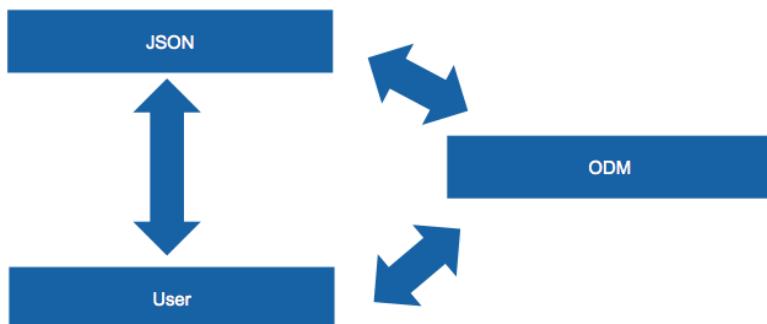
```
db.findOne({_id: id}, function (err, doc) {
  callback( err, doc);
});
```

### Updaten

Es gibt viele Möglichkeiten ein Update zu machen. Entweder Einzelne Werte ändern, Array von einem „document“ anpassen oder das ganze Objekt ersetzen.

```
db.update({_id: id}, {$set: {"state": "DELETED"}}, {}, function (err, doc) {
  publicGet(id,callback);
});
```

ODM (object document mapping) mit Mongoose



## Template Engine

Ziel ist es das Trennen von Controller und View. Express bietet eine render Methode an: app.render(view, [locals], callback). Die Render-Engine muss zuvor konfiguriert werden:

```
app.set('view engine', 'hbs');
```

Sowie der Pfad der Templates angegeben.

```
app.set('views', __dirname + '/views');
```

Express wird standardmässig mit Jade ausgeliefert. Handelbars lassen sich mit folgendem Befehl nutzen.

```
npm install express-hbs --save
```

## Demo 5 – Session &amp; Security

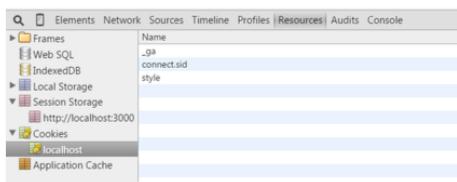
## Cookie

- Details: <https://github.com/expressjs/cookie-parser>
- Installieren
  - npm install cookie-parser –save
- Einbinden
 

```
app.use(require("cookie-parser")());
```
- cookieParser(secret, options)
  - secret: wird zum Signieren benötigt
- Nutzen
  - <http://expressjs.com/api.html#res.cookie>

## Cookie-Tools

- Die meisten Developer Tools bieten die Möglichkeit gespeicherte Cookies anzuzeigen:



## Session

- Session benötigt Cookies
- Installieren
  - npm install session-parser –save
- Einbinden
 

```
app.use(session({ secret: '1234567', resave: false, saveUninitialized: true}));
```
- Nutzen
  - <https://github.com/expressjs/session>
- Sonstiges
  - Beim ersten «Connect» vom Client wird eine Session-Id erstellt und als Cookie zum Client geschickt.
  - Session-Data wird auf dem Server abgespeichert.

## Session

## ■ Was kann man nun damit machen?

- HTTP-Stateless umgehen
  - Widerspricht REST!
  - Login-Status vom User abspeichern
  - Allgemein: Daten Server-seitig einem Benutzer zuordnen
- Tracking

Um REST zu unterstützen muss einerseits das Rückgabeformat geändert werden und anderseits muss es Stateless werden. Also die Session muss durch ein Token ersetzt werden

### Unterschiedliche Formate

```
res.format({
  'text/html': function () {
    res.redirect("/");
  },
  'application/json': function () {
    res.send(true);
  },
});
```

### Token

Das Ziel dahinter ist es einen Stateless Server zu implementieren.

### Idee

Bei jeder Anfrage muss ein Token mitgegeben werden. Im Token sind alle Informationen gespeichert, welche für eine Autorisierung notwendig sind (Ausgestellt an, Ablauf-Datum). Im Token ist eine einmalige zufällige Nummer abgespeichert. In der Datenbank liegt die Information zu wem das Token gehört.

### Vorteil

Jede Anfrage kann zu einem beliebigen Server gesendet werden, solange die Applikation die gleiche Datenbank nutzt.

### Nachteile

Ein Token kann geklaut werden und dadurch würde man dann Zugriff erhalten.

### Wie erhält man das Token

- Token ist bekannt bzw. online sichtbar. Diese Variante wird oft bei REST APIs angeboten
  - o Amazon S3 Storage
  - o Search.ch
- Bei einer Route müssen die persönlichen Daten angeben werden. Falls korrekt, wird ein Token generiert und an den Client geschickt.
  - o Token in die DB speichern, um es auch anderen Servern zugänglich zu machen.

## Motivation

User Experience verbessern durch Auto-Complete in den Formularen und Single-Page-Applikationen.

## Definitionen

Asynchrones Nachladen von Daten sowie Update von der Web Page.

## Nachteile

Der State der Applikation ist nicht bekannt. Er ist schwer zu rekonstruieren. Zudem führen die Such-Maschinen kein JavaScript aus. In den meisten Fällen wird auch der Undo / Redo-Stack vergessen.

- History kann angepasst werden
  - [https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Manipulating\\_the\\_browser\\_history](https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Manipulating_the_browser_history)
- Z.B. `history.pushState({}, "Demo", "bar.html");`
- Sehr oft wird mit URL-Fragment dafür verwendet
  - `history.pushState({}, "demo", window.location.pathname + "#demo3")`
  - Führt zu weiterem Problem
  - Siehe: <https://developers.google.com/webmasters/ajax-crawling/docs/getting-started>

## XMLHttpRequest(XHR)

Um async Request erstellen zu können, wird XMLHttpRequest benötigt. Dies wird native von den Browsern zur Verfügung gestellt. Onreadystatechange wird aufgerufen, falls sich der State ändert.

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (4 === xhr.readyState) {
    if (xhr.status === 200) {
      alert(xhr.responseText);
    } else {
      alert('There was a problem with the request.');
    }
  }
};
xhr.open('GET', url, true); //async
xhr.send();
```

## JQuery.ajax

JQuery bietet verschiedene AJAX-Methoden: get(), getJSON(), post(), ajax()

Es wird Promise verwendet.

```
$.ajax({
  method: "POST", url: "/helloWorld", data: { name: "Michael" }
}).done(function( msg ) {
  alert( "Data Saved: " + msg );
});
```

## Same-Origin-Policy

Es erlaubt XMLHttpRequest nur zur Origin.

## Cross-Origin Resource Sharing

Mechanismus um Cross-Site-Request zu ermöglichen. Der Ziel-Server kann dem Client den Zugriff erlauben. Dies wird vom Browser enforced.

### Fehler falls CORS nicht aktiviert ist

```
XMLHttpRequest cannot load http://localhost:3000/data. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:63342' is therefore not allowed access.
```

**Wichtig:** Chrome erlaubt kein CORS nach localhost

### Aktivieren von Express

```
var allowCrossDomain = function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
  res.header('Access-Control-Allow-Headers', 'Content-Type');
  next();
};

app.use(allowCrossDomain);
```

## Websockets

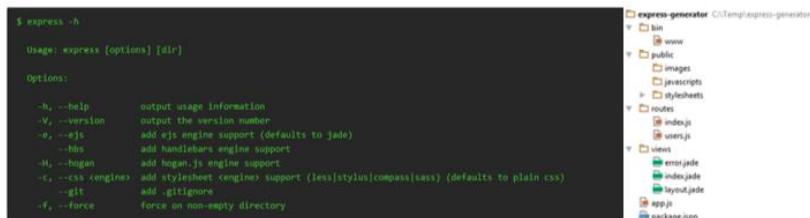
Das klassische Model vom Request-Response hat 2 Probleme. Der Server kann keine Nachricht an den Client schicken und jede Anfrage öffnete eine neue Verbindung. Das Model erschwert es real-time Apps zu machen.

### Ein Workaround war/ist Long-Polling

Der Client öffnet eine Verbindung aber der Server beendet diese nicht. Der Server sendet nun Nachrichten an den Client über diese Verbindung. Probleme sind nun Timeouts und Offene Connections.

Die WebSockets ermöglichen „bi-directional“, „always-on“ Kommunikation. Es hat aber eine komplexe API.

## Generator



Ist sehr gut brauchbar als Start. Der Ordner für Service und Controller fehlt aber und muss noch hinzugefügt werden. Der Generator ist in WebStorm integriert. Es macht eine Trennung zwischen Environment und Server-Bootstrapping.

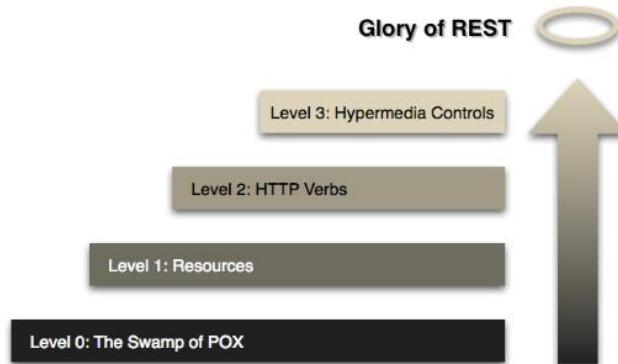
## Deploy

Es kann an verschiedenen Orten genutzt werden. Unteranderem Heruko oder Azure. Grundsätzlich aber überall wo Docker oder NodeJS läuft.

# REST

Was ist Rest

Richardson's Maturity Model



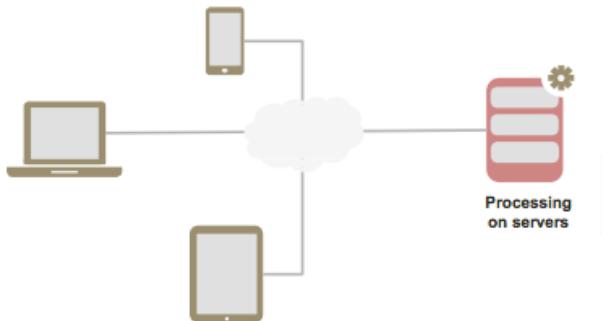
Die Welt vor REST

Zuvor gab es verschiedene Standards und dies von vielen verschiedenen Organisationen. Jedes Ding hat auch so seine Probleme. Schlechte Interoperabilität, jedesmal wurde das Rad neu erfunden und es gab ein Vendor „lock-in“.

Eigenschaften von REST

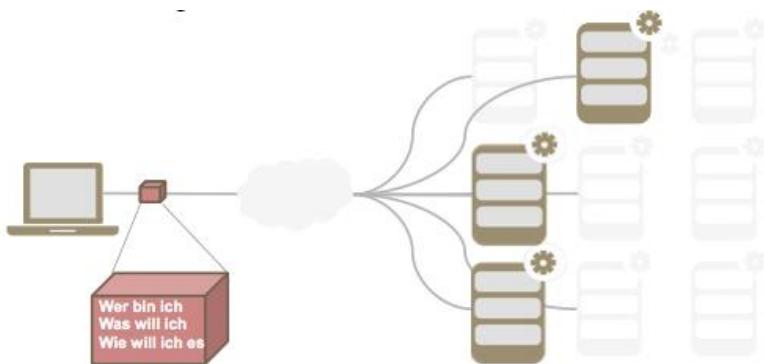
## Client/Server

Trennung von Client und Server-Logik



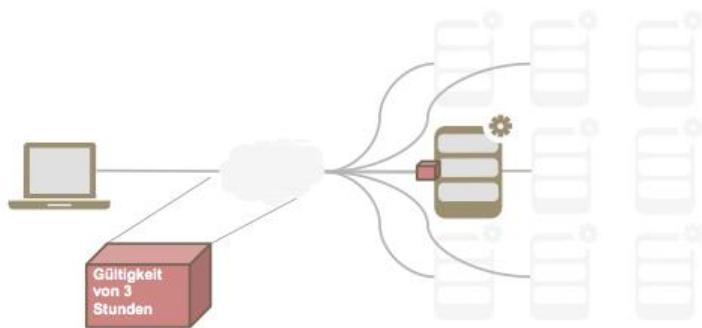
## Statuslose Kommunikation

Jeder Request beinhaltet alle benötigten Informationen. Es ermöglicht eine hohe Skalierung.

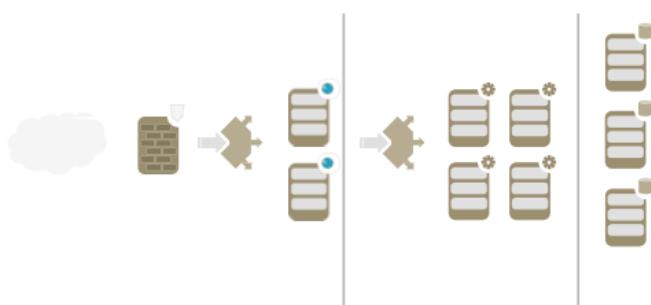


**Cache'bar**

Clients können Antworten zwischenspeichern – insofern dies erlaubt wurde.

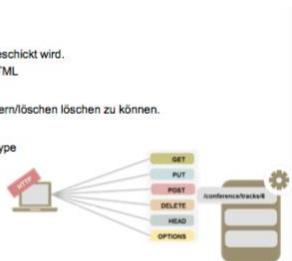
**Layered System**

Erlaubt den Einsatz von Schichtenarchitekturen inkl. Load-Balancers, Proxies und Firewalls.

**Uniform Interface**

Einheitliche Schnittstelle zwischen Client und Server. Es ist selbst beschreibend.

- Identification of resources
  - Die Ressource wird mit der Anfrage identifiziert, z.B. URI
  - Die Ressource ist unabhängig von der Repräsentation welche zum Client geschickt wird.
    - Ermöglicht es unterschiedliche Formate zu wählen z.B. XML / JSON / HTML
- Manipulation of resources through representations
  - Die Antwort soll genügend Informationen beinhalten, um die Ressource ändern/löschen/lösen zu können.
- Self-descriptive messages
  - Die Antwort muss beinhalten wie die Antwort zu behandeln ist, z.B. MIME-Type
- Hypermedia as the engine of application state (HATEOAS)



## Grundprinzipien

**Ressource**

- Alles was genug wichtig ist um eigenständig referenziert zu werden.

**Ressource Name**

- Eindeutige ID der Ressource. Z.B. URI

**Ressource Repräsentation**

- «Dinge» haben mehrere Repräsentationen

**Ressource Links**

- Benutze Hyperlinks als Verknüpfung der «Dinge»

**Ressource Interface**

- Uniform Interface

- Benutze Standard-Methoden

**Kommuniziere statuslos**

## Ressource Name

REST gibt Identifikationen als URI an. Jede Ressource hat eine eindeutige URI.

shop.com/customers/1234/orders

shop.com/products/5849

api.interhome.com/services/status

hsr.ch/employees/mgfeller

usw.

**URL-„Regeln“**

Ressourcen in der Mehrzahl, Query-Parameter nur für Algorithmen / Filter

Gut: /movies?ranking gt 5&type eq action

Schlecht: /orders?customerId=1234

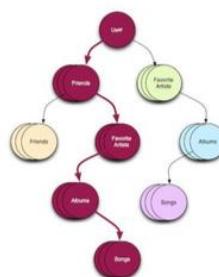
■ Besser: /customers/1234/orders

## Ressource Links

```

1  {
2    "username": "restKungFu",
3    "birthdate": "24.06.1978",
4    "friends": "http://api.myMusicStore.ch/users/restKungFu/friends/",
5    "avatar": "http://api.myMusicStore.ch/users/restKungFu/avatar/320x250.jpg",
6    "state": "registered",
7    "account": [
8      {"mode": "premium",
9       "plan": "Monthly",
10      "type": "creditcard"
11    },
12    "trusted": true
13  }

```



<p><b>GET</b></p> <ul style="list-style-type: none"> <li>Ressource wird angefordert (URI) = read</li> <li>Accept-Header definiert Repräsentation</li> <li>Query-Parameter sind lediglich Filter/Selektoren</li> <li>HTTP Status-Codes beachten</li> </ul> <pre>Request URL: http://api.interhome.com/accommodations/C23940_210.1/?language=de&amp;... Accept: application/json Accept-Encoding: gzip, deflate Host: api.interhome.com User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36 ...</pre>	<p><b>POST</b></p> <ul style="list-style-type: none"> <li>Erzeugt eine neue Ressource             <ul style="list-style-type: none"> <li>Server ist führend für die URI-Generierung (im Gegensatz zu PUT)</li> </ul> </li> <li>Content-Type-Header angeben</li> <li>Wenn eine Ressource erzeugt wurde, wird im Response-Header «Location» die ID (URI) der Ressource mitgeteilt</li> <li>Status Codes: Conflict, Created, No Content, usw.</li> <li>POST's werden nie gecached!</li> </ul> <pre>POST api/orders/ BODY: { productId : 1234, amount: 2 }  Content-Type:application/json; charset=utf-8 Location: http://localhost:25211/api/orders/3 ...</pre>
<p><b>PUT</b></p> <ul style="list-style-type: none"> <li>Aktualisiert / erzeugt eine Ressource             <ul style="list-style-type: none"> <li>keine partiellen Updates.</li> </ul> </li> <li>URI wird von dem Client definiert (im Gegensatz zu POST)</li> <li>Status-Codes: Ok, No Content, Not Implemented, Created</li> <li>PUTs sind idempotent</li> <li>PUTs werden nie gecached!</li> </ul>	<p><b>DELETE</b></p> <ul style="list-style-type: none"> <li>Löscht eine Ressource</li> <li>Status-Codes: Ok, Accepted, No Content</li> <li>DELETE's werden nie gecached</li> </ul>
<p><b>HEAD</b></p> <ul style="list-style-type: none"> <li>Wird genau gleich angewendet wie GET, jedoch ohne die eigentliche Ressource zu erhalten</li> <li>Wird für Caching genutzt</li> <li>Header Informationen und Status-Codes sind relevant</li> </ul>	<p><b>OPTIONS</b></p> <ul style="list-style-type: none"> <li>Gibt an, wie die Ressource verwendet werden darf z.B. welche HTTP-Methoden erlaubt sind</li> <pre>... Server: Apache/2.4.1 (Unix) OpenSSL/1.0.0g Allow: GET,HEAD,POST,OPTIONS,TRACE Content-Type: httpd/unix-directory ...</pre> <li>Wird vom Chrome genutzt ob die CORS Headers gültig sind.</li> <li>Hinweis: Wird selten zu Verfügung gestellt.</li> </ul>
<p><b>PATCH</b></p> <ul style="list-style-type: none"> <li>Wird für partielle Updates einer Ressource genutzt.</li> </ul> <pre>PATCH /order/1 Body: {     status: "versendet" } </pre> <p>Wichtig: Ein Patch muss atomar ablaufen. Ein GET Request sollte nie eine halbe Antwort erhalten.</p>	<p><b>PUT vs. PUT</b></p> <p>POST api/orders/</p> <pre>BODY: { productId : 1234, amount: 2 }</pre> <ul style="list-style-type: none"> <li>Erzeugt eine neue Bestellung</li> <li>Server wählt die ID aus und returniert die erzeugte URL im Location-Header von der Response             <ul style="list-style-type: none"> <li>Location: http://localhost:25211/api/orders/3</li> </ul> </li> </ul> <p>PUT api/orders/3</p> <pre>BODY: { productId : 1234, amount: 2 }</pre> <ul style="list-style-type: none"> <li>Falls nicht vorhanden wird eine neue Bestellung erzeugt mit der id 3...</li> <li>...falls vorhanden wird die vorhandene Bestellung komplett überschrieben – falls erlaubt</li> </ul>

## Ressource Repräsentation

Gleiche Ressourcen haben verschiedene Repräsentationen. HTML, JSON, XML, CSV und so weiter. Die Accept Request-Header definieren die Repräsentation.

## Hateoas (Hypermedia as the engine of application state)

- Prozessgedanke in der Ressource
- Media-Typen beschreiben die Ressource
- Aktionen werden ausgeführt beim folgen von Links
- Jede Antwort beinhaltet den «Application State»
- Selbstbeschreibende API's erzeugen Flexibilität
- Clients können die API «erforschen» ohne Dokumentation und Anleitung

PayPal, Twitter, Github, Stripe oder auch Sharepoint haben diese APIs.

## Vorteile

Es gibt eine Inline Dokumentation. Die API ist explorable. Es sind einfache Client, da die URI's sicher korrekt und aktuell sind. Und die URI kann, falls gewollt, Serverseitig einfach geändert werden.

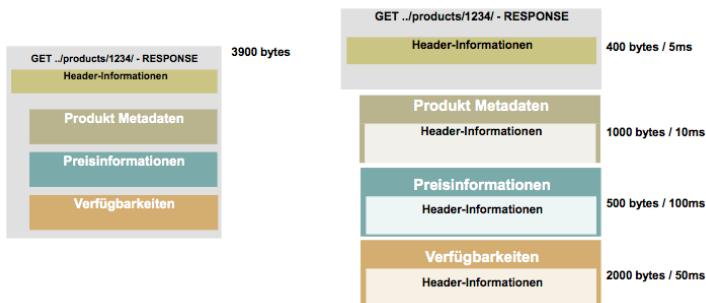
## Nachteil

Sehr Aufwendig auf der Serverseite.

## Optimierungen

Macht es Sinn bei einer Ressource auch die verlinkten Daten mitzugeben? Es kommt ganz darauf an. Wenn die zusätzlichen Daten gar nicht nötig sind, können sie auch weggelassen werden.

## Beispiel



## Versionierung

### ■ Variante 1

- Media-Type = Versionsnummer
- Accept: application/vnd.hsr.avt-v2+json

### ■ Variante 2

- Media-Type verändern sich nicht, ein Qualifier wird hinzugefügt
- Accept: application/vnd.hsr.avt+json;v=2

### ■ Version 3

- Version wird in der URI gesetzt
- GET /api/v2/avt/students/6

### ■ Welche ist die Beste?

- Zwischen 1 und 2 liegen feine Unterschiede (MediaType/ Format), meist eine Konzept-, Implementierungs- und Wartungsfrage.
- 3. Ist die verbreiteste Variante – verletzt aber das Prinzip das eine Ressource genau eine URI besitzt – Version hat mit der Repräsentation zu tun und nicht mit dem «Ding» als solches
- 3. Ist für die Versionierung der API bzw. der Ressource für sich verantwortlich. Im Normalfall möchte man die MediaTypen und dessen Formate und nicht die API versionieren
- Das Beste? 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 and query parameters should not alter the state
  - PUT is Idempotent, POST is not Idempotent
  - ...

- Die meisten Guidelines widersprechen sich in einigen Punkten.
- Die meisten Widersprüche basieren auf der Diskussion wie "Restful" eine Lösung ist.
- Beispiel: <https://www.troyhunt.com/your-api-versioning-is-wrong-which-is/>



- Wichtig:
  - Entscheide Begründung und konsequent anwenden
  - API und Änderungen gut dokumentiert

# Flexibles Layout mit CSS & Flexbox

## Motivation

Heutzutage gibt es hunderte von unterschiedlichen Größen. Die Webseiten sollten sich daran anpassen. Aus Statistiken lassen sich lesen, dass je länger je mehr Traffic über Mobilegeräte geht und nicht mehr über den PC. Und dadurch haben sich auch die Displayverhältnisse geändert. Zusammenfassend:

- Wir wissen nicht, auf welchem Device der User unsere Seite besuchen wird.
- Wir möchten die „User Experience“ trotzdem für jeden User optimieren.
- Wir möchten nicht von der „Bildschirmgröße“ auf die Bedürfnisse des Users schließen (z.B. weniger Infos anzeigen).
- Wir möchten nicht mehrere separate Seiten pflegen müssen.
- Wir möchten ein gutes Google Ranking.

<http://www.awwwards.com/50-examples-of-responsive-web-design.html>



## Flexibles vs. Responsives Layout

### Layout

#### Definition

Die Art und Weise, wie Text und Bilder in einer Zeitung, einer Zeitschrift oder einem Buch angeordnet sind.

#### Informatik (Web)

Bezeichnet wie der Platz (Inhalte) logisch aufgeteilt sind (Header, Content, Footer) und dynamisch auf dem Bildschirm (im Browser-Fenster) platziert werden.

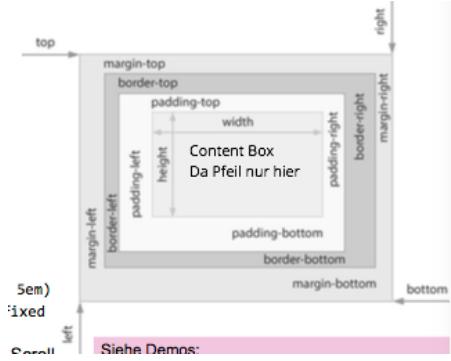
#### Flexibles Layout

Dynamisches (größenadaptives) Layout, welches sich ohne Media-Queris umsetzen lassen. Alternative Begriffe sind Flexibles Design, Adaptive Layouts/Design, Fluides Layout/Design.

#### Responsives Layout

Dynamisches Layout welches für unterschiedliche Geräte, Bereiche von Display-Größen und unterschiedliche Medien separate Layouts definiert. Jedes dieser separaten Layouts kann ein flexibles Layout sein. D.h. Responsive Layouts beinhalten meist auch flexible Layouts. Die Umsetzung mit Media Queries folgt später in der Zusammenfassung.

## CSS Box Model



Es gibt verschiedene Box-Sizing Modelle. Diese geben an worauf sich width und height beziehen. Dazu zählen Border-Box sowie auch Content-Box. Nicht unterstützt wird die Margin Box.

Das Modell erlaubt Definitionen von Breite, Höhe und der Position z.B:

- Height: 100vh // % der View-Height
  - Width: 20% // %-Werte beziehen sich jeweils auf den Parent.
- Berechnen von Werten ist mit calc(100vw - 5em) möglich.
- Bottom: 4em // nur möglich bei position: absolute oder fixed.

## Scrolling beim CSS Box Model

Ist overflow: scroll eines Elements gesetzt wird der Scroll-Bar sichtbar wenn ein Unter-Element (oder Text) aus der Box fällt. Dies gilt auch für Unter-Elemente mit visibility: hidden, aber nicht für Elemente mit display: none.

Mit overflow: hidden wird der Text oder Unterelemente die aus der Box fallen nicht mehr dargestellt (geclipped).

Liegt ein Teil eines Elements unter einem position: absolute positionierten Elementen, so kann durch Setzen einer angemessenen Margin sicher gestellt werden, dass unter diesem überdeckenden Element hervorgescrolled werden kann.

## Werte von Positionen

### position: absolute

Elemente werden aus dem Element-Fluss entfernt. Es kann genutzt werden um ein beliebiges Element absolut zu positionieren mit top, left, bottom, right, width, height. Die Werte werden in px, em, %, vh oder vw angegeben. Sie sind relativ zum ersten Parent mit position: relative oder absolute. Eine Überlappung von Elementen ist erlaubt. Wichtig: Die Position ist relativ zur Seite (html Element). Die absolut positionierten Elemente bewegen sich daher bei Scrollen.

### position: fixed

Kann genutzt werden um ein Element fix an einem Ort auf dem Screen zu platzieren. Zum Beispiel ein Menü welches immer sichtbar sein sollte. Die weiteren Eigenschaften sind wie absolute.

### position: relative

Referenz für Kind-Elemente welche mit position: absolute positioniert werden sollen.

### position: static

Default Wert → Element ist im Fluss

## Flexibles Layout mit absoluter Positionierung

Beispiel (gut)

Die Seite zeigt fünf Absolut positionierte Boxen (box1 bis box5) auf einem weißen Hintergrund. box1 ist eine grüne Box am oberen Rand. box2 ist eine orangefarbene Box mit Text. box3 ist eine rote Box am unteren Rand. box4 ist eine blaue Box in der Mitte links. box5 ist eine gelbe Box in der Mitte rechts.

```

.box {
    box-sizing: border-box;
    border: black solid 1px;
    position: absolute;
    text-align: center;
}

.box1 {
    top: 0;
    left: 0;
    right: 0;
    line-height: 100px;
    background-color: aqua;
}

.box2 {
    top: 100px;
    bottom: 100px;
    left: 0;
    width: 100px;
    line-height: calc(100vh - 200px);
    background-color: darkseagreen;
}

.box3 {
    top: 100px;
    bottom: 100px;
    right: 0;
    left: 100px;
    background-color: gold;
    overflow: scroll;
}

.box4 {
    top: 100px;
    bottom: 100px;
    right: 0;
    width: 100px;
    line-height: 100px;
    background-color: lightcoral;
}

.box5 {
    top: 100px;
    bottom: 100px;
    right: 100px;
    left: 100px;
    background-color: antiquewhite;
}

```

```

<div class="box box1">box1</div>
<div class="box box2">box2</div>
<div class="box box3">box3</div>
<div class="box box4">box4</div>
<div class="box box5">box5</div>

```

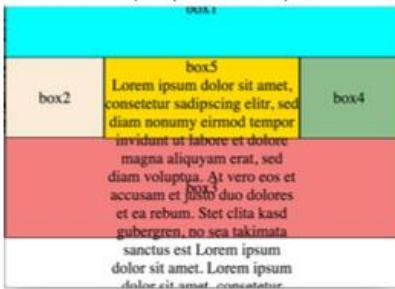
Der CSS-Code definiert die Positionierung und Aussehen der Boxen. box1 bis box4 haben eine Breite von 100px und eine Höhe von 100px. box5 hat eine Breite von 100px und eine Höhe von 100px. Die Farben der Boxen sind unterschiedlich gewählt: aqua, darkseagreen, gold, lightcoral und antiquewhite.

## Web Engineering + Design 2

Dies ist gut, wenn Inhalte (z.B. UI) auf einer Seite mit +/- bekannter Grösse und Font-Grösse passen.

Probleme entstehen wenn das Browser-Fenster zu klein ist für den Inhalt (Fenster/Gerät zu klein, Font zu gross, Text-Inhalte zu lang).

### Demo 05 Beispiel (mit Problem)



## Abhilfe

- Absolute Position mit (Inline)-Blöcken „im Flow“ kombinieren. Die Herausforderung aber besteht bei unterschiedlich langen Spalten.
- Manche Blöcke fix am Viewport positionieren mit position:fixed.
- Zum Teil kann absolute Positionierung relativ zu einem Container helfen (Container mit position:relative | absolute | fixed).

Eigenschaften von Block / Inline / Inline-Block Elementen

### Inline (span, a oder display:inline)

Erlaubt padding und margin auf left und right aber nicht auf top und bottom. Es ignoriert die Höhe und die Grösse, da es sich um einen Textwurm handelt. Erlaubt andere Elemente auf der gleichen Zeile. White-Spaces zwischen Inline Elementen werden dargestellt (Space).

### Block (h1, div, form, p oder display:block)

Erlaubt margin und padding. Jedes Element wird auf einer neuen „Zeile“ (Umbruch) dargestellt. Zudem ist es erlaubt Text-Inhalte zu scrollen, clippen mit overflow: scroll | hidden | invisible.

### Inline-Block (Inline-Flex oder Inline-Table oder display:inline-block)

Erlaubt margin und padding- Zudem erlaubt es anderen Elementen auf der gleichen Zeile mit alignment:vertical-align: top. Die Definition der Höhe und Breite ist möglich. White-Spaces zwischen den Inline-Block Elementen werden dargestellt.

Aktuell wichtigste CSS Layout Technik

### Definition

Die Spezifikation beschreibt ein CSS-Boxmodell, das für das Design der Benutzeroberfläche optimiert ist. Im Flex-Layout-Modell können die Kinder eines flex-Containers in beliebiger Richtung ausgelegt werden und können ihre Größe "flex", entweder wachsen, um ungenutzten Platz zu füllen oder schrumpfen, um ein Überlaufen der Eltern zu vermeiden. Sowohl horizontale als auch vertikale Ausrichtung der Kinder können leicht manipuliert werden. Das Schachteln dieser Kästen (horizontal innen vertikal oder vertikal innerhalb horizontal) kann verwendet werden, um Layouts in zwei Dimensionen zu bauen.

### Flexbox – Display

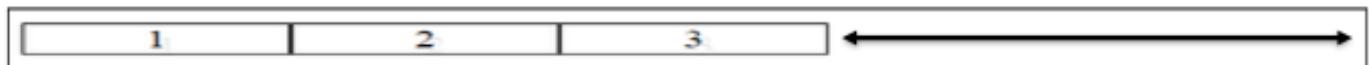
Das Flex-Layout kann auf dem Container wie folgt aktiviert werden. Entweder mit display:flex oder mit display:flex-inline.

```
<div style="display: flex">
</div>
```

Ein Container mit `display:flex` verhält sich wie ein Block Element mit `display:inline-flex` wie ein inline Block.  
Alle Flex-Items verhalten sich wie „inline blocks“.

### Flexbox – Size

Problem – Was passiert mit dem „leeren“ Platz



#### **flex-grow:1**

Entspricht dem Verhältnis wie der „leere“ Platz verteilt wird.

#### **flex-shrink:1**

Entspricht dem Verhältnis wie die Elemente kleiner werden wenn zu wenig Platz vorhanden ist.

#### **flex-basis: 100px/10%**

Entspricht der gewünschten Grösse des Elements. Wenn es weggelassen wird, wird die Grösse des Inhalts genommen.

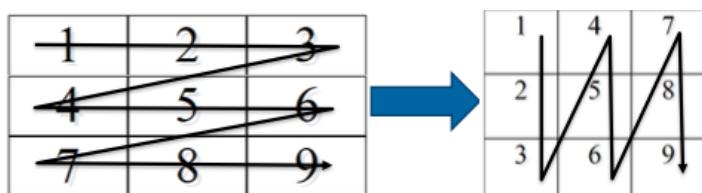
### Befehle

Flex: [flex-grow] [flex-shrink] [flex-basis]

justify-content : flex-start | flex-end | center | space-between | space-around

### Flexbox – Direction

Problem – Leserichtung ist falsch.



#### **flex-direction: row | row-reverse | column | column-reverse**

Ändert die Achsen

#### **flex-direction: column**

Der Container benötigt eine Höhe

Achtung: Die Selektoren gehen auf die „source“ Order.

### Flexbox – Wrap

Problem – Zu viele Elemente auf einer Line



#### **flex-wrap:wrap**

Der Flex-Container „wrapt“ die Elemente sinnvoll. So wird aus der obigen Reihe ein Taschenrechnerfeld. Beachte aber flex-basis und width.

Jede „Row“ ist **unabhängig** mit flex-grow und flex-shrink

## Problem – Elemente sind am falschen Ort

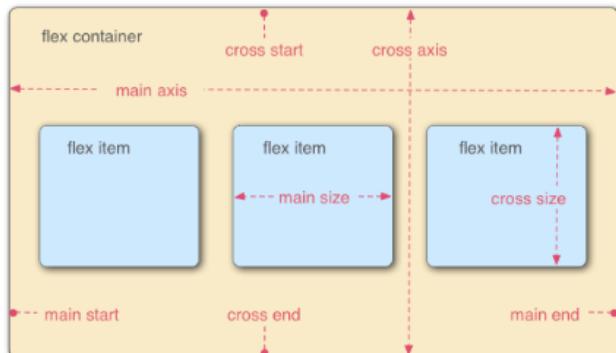
1	2	3
4	5	6
7	8	9
0		

Flex Elemente werden nach der „source“ oder plaziert. Mit order kann die Reihenfolge angepasst werden.

Aber Achtung, es kann zu negativen Nebeneffekten kommen. Default ist es auf „0“-

Die Selektoren gehen auch hier auf die „Source“ Order.

## Flexbox – Direction



Flexbox versucht den Platz vertikal und horizontal auszufüllen.

An der Main-Axis geschieht dies mit Flex-Grow und flex shrink. Wenn die Faktoren 0 sind, findet es nicht statt.

An der Cross-Axis findet dies mit align-items (Für alle Flex-Items, flex-start|flex-end|center|baseline|stretch, default ist stretch) statt.

Zudem ist es auch möglich mit align-self (Wie align-items aber nur für ein Item).

## Browser Support

Broken up by "version" of flexbox:

- (new) means the recent syntax from the specification (e.g. `display: flex;` )
- (tweener) means an odd unofficial syntax from 2011 (e.g. `display: flexbox;` )
- (old) means the old syntax from 2009 (e.g. `display: box;` )

Chrome	Safari	Firefox	Opera	IE	Android	iOS
21+ (new) 20- (old)	3.1+ (old) 6.1+ (new)	2-21 (old) 22+ (new)	12.1+ (new)	10 (tweener) 11+ (new)	2.1+ (old) 4.4+ (new)	3.2+ (old) 7.1+ (new)

Das heisst alten und neuen Syntax mixen.

## Einschränkung von Layouting mit Flexbox

„Die Schachtelung dieser Kästen (horizontal innen vertikal oder vertikal innerhalb horizontal) kann verwendet werden, um Layouts in zwei Dimensionen zu bauen.“

Dieses Layout würde verschachtelte Flexboxen benötigen. Die Globale Flexbox mit Direction = Column. Jede Row wäre eine eigene Flexbox. Das Problem besteht darin, dass kein Flex-Grow auf der Cross-Axis einstellbar ist.





**Idee:** Im einfachsten Use kann das float-Property verwendet um Text um Bilder zu wrappen.

Floats werden/wurden verwendet für das Layouting. Dies führt teilweise zu „komischem“ Behavior.



Details (kein Lernstoff)

## Float

Lässt andere Elemente um ein mit float formatiertes Element herumfliessen. (none, right und left).

## Clear

Die Angabe clear beendet das Umfliessen anderer Elemente. Ein mit clear formatiertes Element ist das Erste, dass nicht mehr neben anderen Elementen steht. Es kann aber trotzdem mit der float-Eigenschaft formatiert werden. damit nachfolgende Elemente um dieses Element wieder herum fliessen. (none, both, left und right)

## Weitere Technik zur Positionierung / Layout: Display Table / Table-Cell

Kein Lernstoff

<ul style="list-style-type: none"> <li>■ <a href="https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Anzeige/display/Tabelle">https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Anzeige/display/Tabelle</a></li> <li>■ <a href="http://colintoh.com/blog/display-table-anti-hero">http://colintoh.com/blog/display-table-anti-hero</a></li> <li>■ <a href="https://www.sitepoint.com/solving-layout-problems-css-table-property">https://www.sitepoint.com/solving-layout-problems-css-table-property</a></li> <li>■ <a href="http://codepen.io/backflip/pen/edHJi">http://codepen.io/backflip/pen/edHJi</a> / <a href="http://codepen.io/backflip/pen/cCFth">http://codepen.io/backflip/pen/cCFth</a></li> </ul>	Unter Nutzung von Informationen von Thomas Jaggi / Matthias Meier HSR CAS FEE
<ul style="list-style-type: none"> <li>■ <b>Bewertung:</b> <ul style="list-style-type: none"> <li>■ Eigentlich zur Nutzung für Responsive Table Layout (display: table und display table-cell)</li> <li>■ Mächtige Technik, aber selten eingesetzt zum generellen Layout</li> <li>■ Problem: Nicht ganz vollständige Standardisierung: z.B. <a href="http://www.w3.org/TR/CSS21/visuren.html#choose-position">http://www.w3.org/TR/CSS21/visuren.html#choose-position</a> «The effect of 'position:relative' on ... table-cell ... elements is undefined.» -&gt; Elemente mit "display: table-cell" können in Firefox nicht relativ positioniert werden</li> </ul> </li> </ul>	

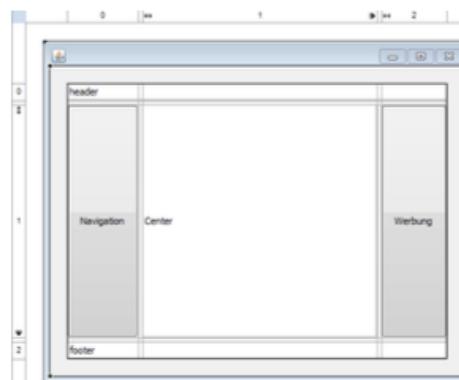
## Weitere Technik zur Positionierung / Layout: CSS Grid (Ausblick)

Kein Lernstoff

<https://css-tricks.com/snippets/css/complete-guide-grid/>

### Idee:

- Klassisches Desktop Layout zu Verfügung stellen
- Grid definieren mit Anzahl Rows / Columns
  - Verteilung der Größen / Verhältnisse auf beiden Achsen möglich



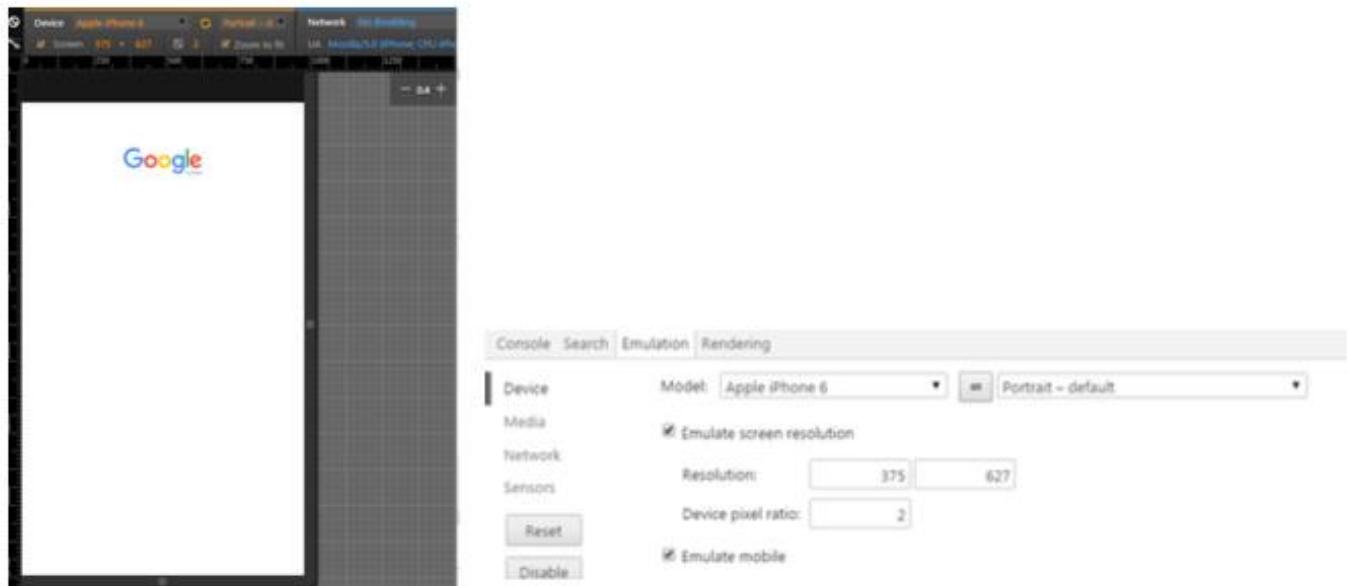
## Weitere Praxistips: Zentrieren von Elementen (horizontal & vertikal)

Kein Lernstoff

## Layout Testing

Am einfachsten lässt sich dies Testen, wenn man den Fenster grösser oder kleiner zieht.

## Chrome Debug Tools



## Emulatoren und Simulatoren

iOS-Simulator (funktioniert auf IE11 aber nicht)

[https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS\\_Simulator\\_Guide/Introduction/\\_Introduction.html](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/_Introduction.html)

## Android Emulator

<http://developer.android.com/tools/help/emulator.html>

## Windows Phone Emulator

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402563\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402563(v=vs.105).aspx)

## Hardware

Reale Devices sind immer noch unverzichtbar. Einerseits aufgrund der Performance anderseits aber aufgrund der Physischen Interaktion („Responsiveness“, Grösse und Erreichbarkeit von interaktiven Elementen, etc...).

Dazu gibt es öffentliche Device Lab wie [opendevicelab.ch](http://opendevicelab.ch).

## Browser Support: Progressive Enhancement & Graceful Degradation

Im Web werden nie alle Browser auf dem selben technischen Stand sein. Daher stellt dies sich die Frage wie wir damit umgehen. Es gibt dazu drei Ansätze.

### Graceful Degradation

Es werden alle modernen Features genutzt. Wenn Sie nicht vorhanden sind, sollen Fallbacks eingesetzt werden, dem Nutzer eine sinnvolle Alternative angeboten werden oder auf das „Problem“ hingewiesen werden.

## Beispiel

```
<p id="printthis">
  <a href="javascript:window.print()">Print this page</a>
</p>
```

### Notwendig für dieses Feature

- **JavaScript**

- **window.print()**

### Mögliche «Graceful Degradation»:

```
<noscript>
  <p class="scriptwarning">
    Print a copy of your confirmation.
    Select the "Print" icon in your browser,
    or select "Print" from the "File" menu.
  </p>
</noscript>
```

## Progressive Enhancement

Inhalt soll für alle Browser zugänglich sein. Gestart wird mit der Grundfunktionalität (Annahme: kein JS, keine Media Queries). Zusatzfunktionalität mit zusätzlichem CSS und unobtrusiven Javascript.

```
<p id="printthis">Thank you for your order. Please print this page for your records.</p>
```

### Notwendig für dieses Feature

- **Nichts**

### Mögliche «Progressive Enhancement»:

```
<p id="printthis">Thank you for your order. Please print this page for your records.</p>
<script type="text/javascript">
(function(){
  if(document.getElementById){
    var pt = document.getElementById('printthis');
    if(pt && typeof window.print === 'function'){
      var but = document.createElement('input');
      but.setAttribute('type','button');
      but.setAttribute('value','Print this now');
      but.onclick = function(){
        window.print();
      };
      pt.appendChild(but);
    }
  }
})();
</script>
```

<https://jqueryui.com/accordion/>

#### Mit JavaScript



#### Ohne JavaScript

Section 1  
Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus mi felis, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.  
Section 2  
Sed non urna. Donec et ante. Phasellus eu ligula. Vestibulum sit amet purus. Vivamus hendrerit, dolor at aliquet laoreet, mauris turpis porttitor vel, felis. Nascetur interdum tellus libero ac justo. Vivamus non quam. In semper feugiat urna.  
Section 3  
Nam enim risus, molestie et, porta ac, aliquam ac, risus. Quisque lobortis. Phasellus pretium purus in mollis. Aliquam erat volutpat. Praesent libet et tellus pulvinar tempus. Sed ac felis. Sed commodo, magna quis lacina ornare, quam ante aliquam nisi, eu facilisis leo purus venenatis dui.



### Wann sollte man welchen Ansatz wählen

- **Graceful Degradation**

- Prototyp erstellen
- Altes Projekt weiter entwickeln
- Keine Zeit / Budget
- Die Webseite wäre ohne nicht lauffähig z.B. Bildbearbeitungs-Tools

- **Progressive Enhancement**

- Die Webseite soll auf jeden Device nutzbar sein
- Neue Funktionalität kann hinzugefügt werden ohne den «Core» zu ändern
- Bei Vorhandensein von modernen Features werden diese nachträglich hinzugeführt

Fazit: «Graceful Degradation» ist günstiger für die Entwicklung aber normalerweise teurer als «Progressive Enhancement» in der Wartung.

#### Mit JavaScript

```
<div id="accordion" class="ui-accordion ui-widget ui-helper-reset" role="tablist">
  ><h3 class="ui-accordion-header ui-state-default ui-accordion-header-active ui-state-active ui-corner-top ui-accordion-icons" role="tab" id="ui-id-1" aria-controls="ui-id-2" aria-selected="true" aria-expanded="true" tabindex="0">...</h3>
```

#### Ohne JavaScript

```
<div id="accordion">
  <h3>Section 1</h3>
  <div>...</div>
  <h3>Section 2</h3>
  <div>...</div>
  <h3>Section 3</h3>
```



### Wie erkennt man ob ein Browser ein «Feature» besitzt

- Variante 1: Browser Version überprüfen:

```
if (browser === "the-one-they-make-you-use-at-work") {
  getTheOldLameExperience();
} else {
  showOffAwesomeNewFeature();
}
```

- Variante 2: Feature detektieren:

- 'webkitSpeechRecognition' in window

- Variante 3: Modernizr

- Plugin Model um verschiedene Feature zum detektieren – verwendet dazu Variante 2
- Fügt CSS-Classes dem html Tag hinzu z.B. no-speechsynthesis bzw. speechsynthesis
- Einheitliche API
- <https://modernizr.com/>

Fazit: Variante 3 ist in den meisten Fällen vorzuziehen.

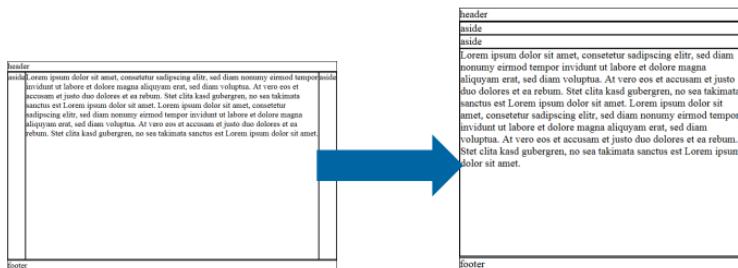
## Mobile First

Ist eine Art von „Progressive Enhancement“.

# Responsive Layout mit CSS Media-Queries

## Responsive Layout mit Media Queries

Wie im letzten Kapitel festgestellt, hat das fluide Layout seine Grenzen. Das untere Bild auf der linken Seite zeigt dies sehr gut.



Das obige Beispiel hat folgenden Code. Aber Achtung: Dies ist nicht «Mobile First».

```
body{
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}

header, footer {
  flex: 1 1 100%;
}

main{
  flex: 1 1 0;
}

<header>header</header>
<nav>nav</nav>
<main>Lorem ipsum ...</main>
<aside>aside</aside>
<footer>footer</footer>
```

Der Sinn hinter Media Queries ist es spezifische CSS für unterschiedliche «Medien» zu nutzen. Dies lässt sich anhand der Zustände, Typen, Dimensionen sowie diversem Sonstigen realisieren.

<ul style="list-style-type: none"> <li>■ <b>@media screen {</b></li> <li>■ <b>@media print {</b></li> </ul>	<b>@media ([width min-width max-width]: 375px) {}</b> <b>@media ([height min-height max-height]: 667px) {}</b> <b>@media ((device-width min-device-width max-device-width]: 375px) {}</b> <b>@media ((device-height min-device-height max-device-height]: 667px) {}</b>
<b>@media (orientation: landscape) {}</b>	<b>@media (min-resolution: 300dpi) {}</b> <b>@media (min-color: 1) {}</b>

In den Queries dürfen auch Operatoren verwendet werden um mehreres gleichzeitig zu erreichen. Die Operatoren sind and, only und not.

```
@media (min-width: 20em) and (max-width: 30em) {}

@media (max-width: 10em), (min-width: 20em) and (max-width: 30em), (min-width: 40em) {}

@media not screen {}

@media not screen and (min-width: 20em) {}

@media only screen {}
```

## Einheiten

Die Einheiten sind gleich, wie an anderen Orten. Darunter gehören px, em, rem %, etc. Die relativen Einheiten sind relativ zum «initialen Wert» (Beispiel: 10em = 10 \* «initial font size» = 10 \* 16px).

Wenn immer möglich sollte «em» verwendet werden für eine höchst mögliche Accessibility. Grund dafür ist, dass die Default-Font-Size im Browser berücksichtigt wird.

### Folgende typische Trigger Punkte für die Media Queries sind vorhanden:

- 480px / 30em – Smartphone
- 768 / 48em – Tablets
- 992 px / 62em – Desktops

Media-Attribut in link Referenzen auf externe Stylesheets

Externe Stylesheets die mit <link rel=>stylesheet href=>appStyles.css im Head des Dokumentes eingebunden werden, müssen vollständig geladen sein bevor das Rendering der Seite beginnt.

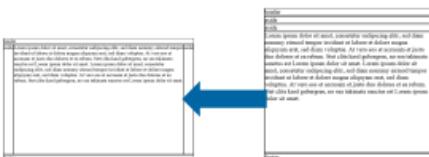
Link Referenzen erlauben auch die Spezifikation eines Media-Attributes z.B. <link rel="stylesheet" href="LargeScreenLayout.css" media="(min-width: 30em)">.

Ist ein Media-Attribut spezifiziert, so wird evaluiert ob die Eigenschaft zutrifft. Wenn ja, dann wird das Stylesheet regulär geladen und direkt evaluiert. Wenn nicht, dann blockiert das Laden des Files nicht das Rendering. Das Stylesheet wird mit tieferer Priorität im Hintergrund geladen. Evaluiert wird das Stylesheet immer erst dann wenn die Bedingung zutrifft.

Achtung, dies ist nur supported in den HTML5 Browsern-.

Mobile First Layout mit Media-Queries

Bei den CSS Definitionen sollte nach Mobile First gearbeitet werden. Das heisst die Grössen für's Mobile definieren und Queries für grössere Bildschirme erstellen und nicht umgekehrt.



```

body{
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}

header, footer {
  flex: 1 1 100%;
}

main{
  flex: 1 1 0;
}

```

```

body{
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}

header,
footer,
aside,
nav {
  flex: 1 1 100%;
}

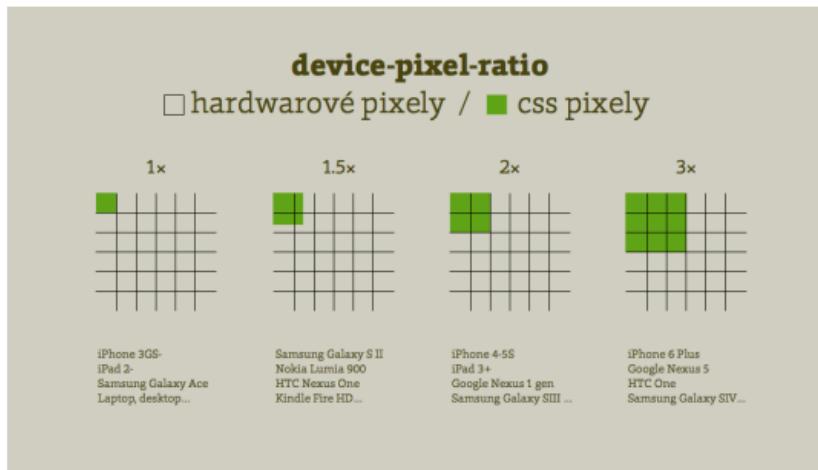
main{
  flex: 1 1 0;
}

@media (min-width: 30em) {
  aside, nav {
    flex: 0 0 0;
  }
}

```

### Prozess der Erstellung von Media Queries

- Start mit Mobile Layout (speichern)
- Dann umbauen zum Desktop Layout (HTML darf nicht angefasst werden)
- Einzeln testen
- Durch Copy & Paste in ein File integrieren (Desktop CSS in die Media Queries)
- Reduktion des CSS in den Media Queries (nur die Änderungen behalten)



Die «Device-Pixel-Ratio definiert das Verhältnis zwischen Device-Pixel und CSS-Pixel.

Dies kann im Browser über `window.devicePixelRatio` abgefragt werden. Bei Zoomen wird dies verändert.

Smartphones haben unterschiedliche «Device-Pixel-Ratio», da Sie unterschiedliche Auflösungen haben.

Abfrage von spezifischen Geräten

Mit Media Queries lässt sich CSS für spezifische Geräte spezifizieren.

```
/* ----- iPhone 6 ----- */
/* Portrait and Landscape */
@media only screen
    and (min-device-width: 375px)
    and (max-device-width: 667px)
    and (-webkit-min-device-pixel-ratio: 2) {
...
}
/* Portrait */
@media only screen
    and (min-device-width: 375px)
    and (max-device-width: 667px)
    and (-webkit-min-device-pixel-ratio: 2)
    and (orientation: portrait) { ... }
```

Weitere «Tipps & Tricks» zu Layout & Media Queries

Bei Texten `max-width:60em` spezifizieren um überlange Zeilen zu vermeiden.

Media-Queries sollten in em angegeben werden wenn Cross-Browser Kompatibilität optimiert werden soll.

Bevorzugt mit width statt mit device-width arbeiten, wenn es um die Breite geht.

Media-Queries können auch in Javascript genutzt werden.

```
if (window.matchMedia("(min-width: 35em)").matches) {
    /* Der Viewport ist mindestens 35em breit */
} else {
    /* Das Medienmerkmal trifft nicht zu, alle anderen Viewports */
}
```

Falls mit JS auf resize oder orientationchange von Fenstern reagiert werden sollte, sollte der entsprechende EventListener auf window registriert werden

```
window.addEventListener("orientationchange",
    function() { alert("the orientation of the device is now " + screen.orientation.angle);
});
```

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

Die Anweisung ist wichtig um die «Intelligenz» mobiler Browser zu unterbinden. So skaliert der Mobile Safari Browser den View defaultmässig auf eine Breite von 964px. Dies ist nicht angemessen für responsive Sites, da z.B. Media Queries nicht greifen.

```
<body>
  <p>Viewport Demo</p>
  <div style="width: 320px;
    height: 10px;
    border: black solid 1px">
    320px</div>
  <div>...langer text....</div>
</body>
```



## Responsive Images

### Problem

Die Seite soll sowohl auf 100-Zoll-Retina-Screens als auch auf einer Armbanduhr gut aussehen. Es kommt zu Problemen mit Restergrafiken.

### Lösungsansätze

- Alle Devices laden das grösstmögliche Bild
  - o Massive Verschwendungen von Daten, miserable Performance
- Es wird eine Kompromiss-Grösse definiert
  - o Mittel-massive Verschwendungen von Daten, mittel-miserable Performance, schlechte Qualität auf grossen Screen
- Alle Devices laden ein optimiertes Bild
  - o So sollte es sein, aber wie erreichen wir dies?

Dabei ist wichtig, da über 50 % des Datenvolumens Bilder sind.

#### Code mit img

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


```

**Src** fallback

**Srcset** Eine Liste von Bildern mit zusätzlicher Information zur Grösse oder der Pixel-Dichte

**Sizes** Gibt an wie Gross das Bild sein sollte im Layout

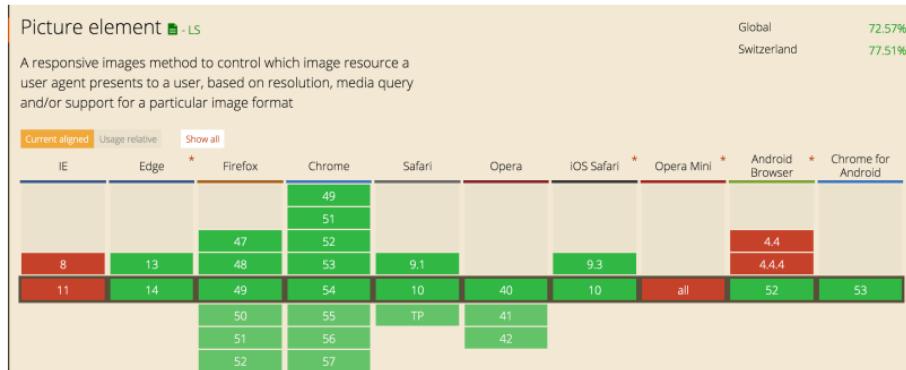
■ sizes=" [media query] [length], [media query] [length] ... etc"

Der Browser wählt dann aus. Es ist ein unterschiedliches Verhalten je nach Browser vorhanden. Zum Beispiel wenn ein «grösseres» Bild schon geladen ist, wird dies beibehalten.

```
<picture>
  <source media="(min-width: 62em)" srcset="Large.png">
  <source media="(min-width: 36em)" srcset="Medium.png">
  <source srcset="Small.png">
  
</picture>
```

Eine Source kann zur Unterscheidung vom «Layout» verwendet werden. (z.B. Hochbild und Breitbild). Jede einzelne Source verhält sich wie ein «img» Tag. Das Media-Query wird nicht optimiert.

## Support



Bei den restlichen Prozenten kommt es zu Polyfill.

## Polyfill

Ein Polyfill (auch Polyfiller / Shim / Fallback) ist ein in Javascript geschriebener Code-Baustein, der in älteren Browsern eine neuere, von diesem nicht unterstützten Funktion mittels eines Workarounds nachrüsten soll.

**Picture:** <https://scottjehl.github.io/picturefill/>

**CSS Grid:** <https://github.com/FremyCompany/css-grid-polyfill>

## Einschub – Performance Tool

### Webpagetest.org

Zeigt auf wie die Webseite über die Zeit geladen wird. Dies findet auf echten Geräten statt. Die Connection-Geschwindigkeit kann angegeben werden.

## CSS Präprozessor

Ein Präprozessor ist ein Computerprogramm, das Eingabedaten vorbereitet und zur weiteren Bearbeitung an ein anderes Programm weitergibt.

## Präprozessoren

- Sass-lang.com
- Lesscss.org
- Lernboost.githhub.io/stylus

## Vorteile / Möglichkeiten

CSS Präprozessoren sind nicht an die Limitationen von CSS gebunden und ermöglichen es uns dadurch, (potentiell) effizienter und besser warbaren Code zu schreiben.

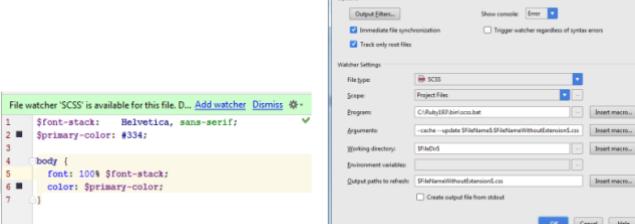
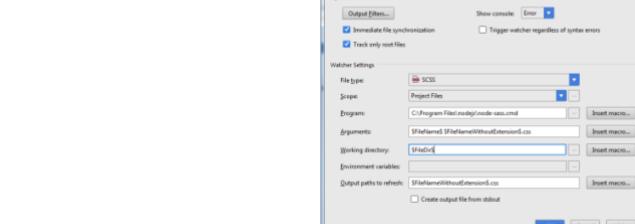
CSS Präprozessoren ermöglichen es Software Engineering Prinzipien im CSS anzuwenden.

- Kein Copy & Paste
- Modularisierung
- Wiederverwenden von Funktionalitäten

## Installation

Sass-Lang.com Dafür ist Ruby notwendig. Er ist der Default File-Watcher für Webstorm

Node-Saas Verfügbar unter [github.com/saas/node-sass](https://github.com/saas/node-sass). Es wird für Gulp verwendet (Nicht Teil der Vorlesung). Es kann über die Command-Line aufgerufen werden oder auch als File-Watcher in WebStorm eingebunden werden.

<p>■ <a href="http://sass-lang.com/install">http://sass-lang.com/install</a></p>  <pre>\$font-stack: Helvetica, sans-serif; \$primary-color: #334; body {   font: 100% \$font-stack;   color: \$primary-color; }</pre> <p>File watcher 'SCSS' is available for this file. D... Add watcher Dimins ⌂</p> <p><b>npm install node-sass -g</b> Console muss Admin Rechte besitzen</p> <p><b>File type:</b> SCSS    <b>Scope:</b> Project Files    <b>Program:</b> C:\Ruby20-x64\bin\nodejs.bat <b>Arguments:</b> --cache --update \$fileNamet\$filenameWithoutExtension.css <b>Working directory:</b> \$fileDir\$ <b>Environment variables:</b> Output paths to refresh: \$fileNamet\$filenameWithoutExtension.css <b>Output paths to refresh:</b> \$fileNamet\$filenameWithoutExtension.css</p>	<p>■ <a href="http://codepen.io/anon/pen/BopWGg">http://codepen.io/anon/pen/BopWGg</a></p>  <p>HTML    CSS    JavaScript    Behavior</p> <p><b>CSS Preprocessor:</b> SCSS</p>
---	---

## SASS vs. SCSS

<p><b>SASS SYNTAX</b></p> <pre>\$font-stack: Helvetica, sans-serif \$primary-color: #334;  body   font: 100% \$font-stack   color: \$primary-color</pre>	<p>Sass    SCSS</p> <p><b>SCSS SYNTAX</b></p> <pre>\$font-stack: Helvetica, sans-serif; \$primary-color: #334;  body {   font: 100% \$font-stack;   color: \$primary-color; }</pre>
--	---

**SASS**

Da spricht man von der "Einrücke»-Syntax. Dies hat historische Gründe. Es ist weniger Schreibarbeit nötig und es ist einfacher zum Aussprechen.

**SCSS**

Es hat die CSS-Syntax mit ; und (). CSS ist mit SCSS kompatibel. Dabei ist aber die Lernkurve kleiner. Es ist aber übersichtlicher, da weniger «wegelassen» wird.

**Fazit** SCSS verwenden

## Features

Spezielle Zeichen – Übersicht

\$	Variablen
&	Referencing Parent Selectors
%	Placeholder
@	Steuerzeichen / Directives z.B. @if @else @function
#{}{}	Interpolation / Einsetzen (Den Value einer Variable einsetzen)
/**/ //	Kommentare, Kommentare können auch mit Sass bearbeitet werden.

## Variablen

Ermöglichen das wiederverwenden von Konstanten Werten wie zum Beispiel von Farben. Sprechende Variablen machen den Code wartbar (keine Magic Numbers). Die Variablennamen müssen mit einem \$ beginnen.

```
$colorMain: #73c92d;

a {
  color: $colorMain;
}

.highlight {
  background-color: $colorMain;
}
```



```
a {
  color: #73c92d;
}

.highlight {
  background-color: #73c92d;
}
```

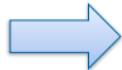
Ermöglicht es die Selektoren zu Strukturieren und zu verschachteln. Es sollte aber nicht ausarten, maximal 3 Ebenen.

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}

nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```



Direct Descendant ist auch möglich.

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  > li {
    display: inline-block;

    > a {
      display: block;
      padding: 6px 12px;
      text-decoration: none;
    }
  }
}

nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav ul > li {
  display: inline-block;
}
nav ul > li > a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```



### Parent-Selektor

& wird durch den Parent Selektor ersetzt. Das & kann überall im Selektor stehen.

```
.headline {
  margin-top: 5em;
  color: black;

  .sidebar & {
    margin-top: 2em;
  }

  &:hover {
    color: darkred;
  }
}

.headline {
  margin-top: 5em;
  color: black;
}
.sidebar .headline {
  margin-top: 2em;
}
.headline:hover {
  color: darkred;
}
```



Es ermöglicht das Auslagern von geteilten Informationen sowie das Aufteilen von Stylesheets auf mehrere Files. Partials müssen mit dem Unterline «\_» beginnen. Die Partials werden von Saas aber nicht in einen separates CSS übersetzt.

Bei der Importanweisung muss die Datei-Endung nicht angegeben werden und das \_ kann weggelassen werden. Die Import-Anweisung erzeugt ein CSS-File.

```
@import 'constants';

h1,h2,h3,h4 {
  font-family: $header-font;
}

h1 {
  color:$brand-color;
}

section {
  h1 {
    color:$accent-color;
  }
}

//file: _constants
$brand-color: rgb(255,0,0);
$accent-color: darken($brand-color, 10);
$body-font: "Helvetica Neue", Arial, sans-serif;
$header-font: "Segoe UI", $body-font;
$width: 960px
```

## Mixins

Ermöglichen das Wiederverwenden von CSS Snippets. @mixin definiert ein neues Mixing. @include lädt das Mixing in das Element.

```
.element {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
}

@mixin visuallyhidden() {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
}
```

## Mixing mit Parameter

Mixing erlaubt auch Parameter.

```
@mixin border-radius($radius: 1em) {
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  -ms-border-radius: $radius;
  border-radius: $radius;
}

.box {
  @include border-radius(1rem);
```

@content wird mit dem Content vom @include abgeführt.

```
@mixin breakpoint($size) {
  @media screen and (min-width: $size) {
    @content;
  }
}

@include breakpoint(30em) {
  .box {
    @include border-radius(0px);
  }
}

@media screen and (min-width: 30em) {
  .box {
    -webkit-border-radius: 0px;
    -moz-border-radius: 0px;
    -ms-border-radius: 0px;
    border-radius: 0px;
  }
}
```

Dabei wird & richtig gesetzt.

```
@mixin hover() {
  &:hover{
    background: red;
  }
}

.box {
  @include hover();
}
```



```
.box:hover {
  background: red;
}
```

## Extend / Inheritance

```
.icon {
  transition: background-color ease .2s;
  margin: 0.5em;
}

.error-icon {
  @extend .icon;
  /* error specific styles... */
}

.info-icon {
  @extend .icon;
  /* info specific styles... */
}
```



```
.icon, .error-icon, .info-icon {
  transition: background-color ease .2s;
  margin: 0.5em;
}

.error-icon {
  /* error specific styles... */
}

.info-icon {
  /* info specific styles... */
}
```

## Mit dem % Placeholder



```
%icon {
  transition: background-color ease .2s;
  margin: 0.5em;
}

.error-icon {
  @extend %icon;
  /* error specific styles... */
}

.info-icon {
  @extend %icon;
  /* info specific styles... */
}
```



```
.error-icon, .info-icon {
  transition: background-color ease .2s;
  margin: 0.5em;
}

.error-icon {
  /* error specific styles... */
}

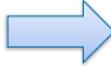
.info-icon {
  /* info specific styles... */
}
```

## Vergleich zwischen Extend/Inheritance Mixing

```
@mixin icon {
  transition: background-color ease .2s;
  margin: 0.5em;
}

.error-icon {
  @include icon;
  /* error specific styles... */
}

.info-icon {
  @include icon;
  /* info specific styles... */
}
```



```
.error-icon {
  transition: background-color ease .2s;
  margin: 0.5em;
  /* error specific styles... */

}

.info-icon {
  transition: background-color ease .2s;
  margin: 0.5em;
  /* info specific styles... */
}
```

## Generell

Alle Varianten haben Vor- und Nachteile. Es ist wichtig, das generierte CSS im Auge zu behalten.

### Mixing

Zu Benutzen falls Parameter benötigt werden. Es generiert weniger CSS-Selektoren, aber ein grösseres CSS-File. Der generierte Code ist hier verständlich und eher leicht lesbar.

### Extends

Generiert weniger CSS Code. Es sollte nur für thematische verwandte Vererbung verwendet werden. Zum Beispiel die Icon-Vererbung.

Im Zweifelsfall sollte Mixing verwendet werden.

## Einheiten

- **Numbers** (z.B. 10px, 1px)
- **Strings**, mit oder ohne "" (z.B. «foo»)
- **Colors** (z.B. blue, #04a3f9)
- **Booleans**
- **Nulls**
- **Listen** – mit Komma oder mit Leerzeichen getrennt (z.B. \$list = 1.5em 1em 0 2em)
- **Maps** (z.B. (key1:value1, key2:value2))

## Einheiten und Rechnen

Die Einheiten werden wie in der Physik gerechnet.

2px + 2px;	4px
2 * 1px	2px
(2 + px) * 2	«Error»
2px / 1px	2
2px + 1mm;	5.77953px
2px * 0	0px
10px * 10px	100px*px => «Error»
10 + px	«10px»
10 + 0px	10px

## Loops

Sie können auch Loops und Verzweigungen innerhalb von Sass verwenden.

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



```
media all and (max-width: 30em) {
  body {
    width: 4px;
  }
}
@media all and (max-width: 46em) {
  body {
    width: 2px;
  }
}
```

## Funktionen

Sass erlaubt es eigene Funktionen zu definieren.

```
$width : 2px;
@function remove($para) {
  @return $para / ($para * 0+1) ;
}
body{
  width: remove($width);
}
```



```
body {
  width: 2;
}
```

# Responsive Web-Design

Kurzrepetition WED1 Usability + User Centered Design

Was ist Usability?

ISO 9241-11 (Effektiv, Effizient, Zufrieden)

## Garrett Ebenen mit besonderer Relevanz bei der Optimierung der Usability

Validierung: Heuristische Analyse und Usability Tests mit Aufgabenstellung und Screen-Flows mit Daten

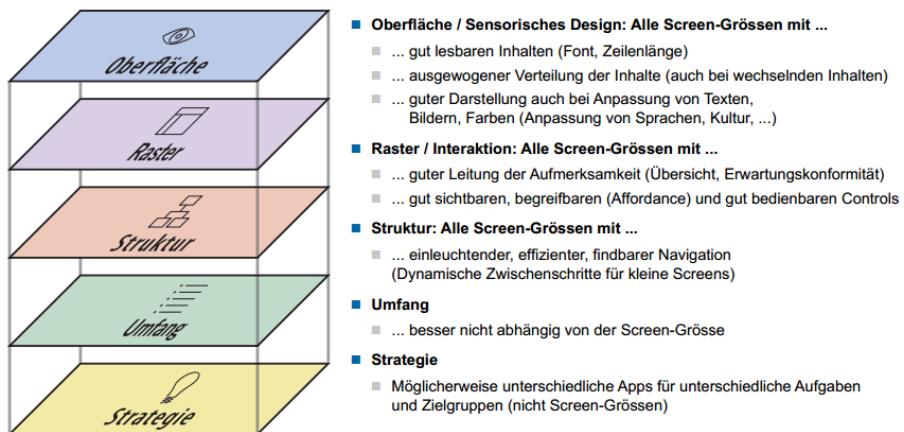
- **Oberfläche / Sensorisches Design**
  - o Beachten: Wahrnehmung nur im Fokus
  - o Beachten: Eingeschränkte Wahrnehmung von Farben und kulturbedingte Interpretation
- **Raster Design mit Interaktionsdesign**
  - o Beachten: Hierarchie von Informationen, Leiten der Aufmerksamkeit des Leseflusses
  - o Affordance von Controls (Begreifbarkeit)
  - o Design Patterns
- **Strukturdesign**
  - o User wissen immer wo sie sind, was sie machen können und was passiert?
  - o Sitemap, Card-Sort

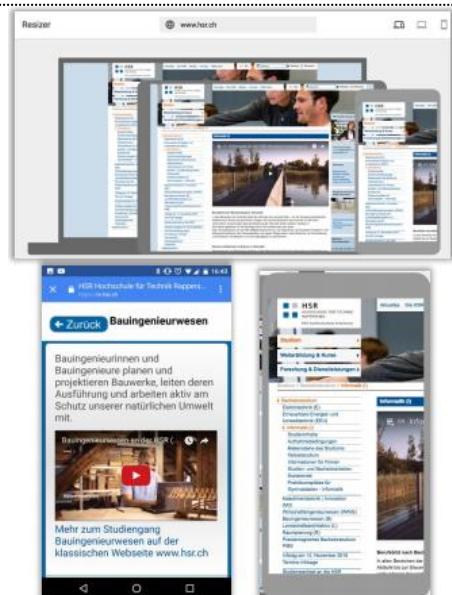
## Garrett Ebenen mit besonderer Relevanz beim User-Centered Design

Validiert durch Beobachtung, Befragung, ....

- **Umfang** (was kann man machen) und Strategie (optimiert für welche Zielgruppe). Definiert ist es durch realistische Szenarios (Kontext, User, Task, Toll) mit Auslöser: Genutzt in Usability Test.

Herausforderungen für Responsive Web-Sites geordnet nach Garret Ebenen





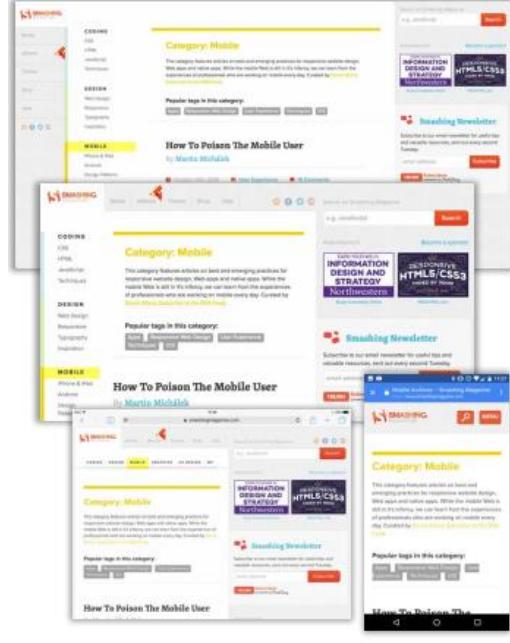
- **Oberfläche / Sensorisches Design:** Alle Screen-Größen mit ...
  - ... gut lesbaren Inhalten (Font, Zeilenlänge)
  - ... ausgewogener Verteilung der Inhalte (auch bei wechselnden Inhalten)
  - ... guter Darstellung auch bei Anpassung von Texten, Bildern, Farben (Anpassung von Sprachen, Kultur, ...)
- **Raster / Interaktion:** Alle Screen-Größen mit ...
  - ... guter Leitung der Aufmerksamkeit (Übersicht, Erwartungskonformität)
  - ... gut sichtbaren, begreifbaren (Affordance) und gut bedienbaren Controls
- **Struktur:** Alle Screen-Größen mit ...
  - ... einleuchtender, effizienter, findbarer Navigation (Dynamische Zwischenschritte für kleine Screens)
- **Umfang**
  - ... besser nicht abhängig von der Screen-Größe
- **Strategie**
  - Möglicherweise unterschiedliche Apps für unterschiedliche Aufgaben und Zielgruppen (nicht Screen-Größen)



- **Oberfläche / Sensorisches Design:** Alle Screen-Größen mit ...
  - ... gut lesbaren Inhalten (Font, Zeilenlänge)
  - ... ausgewogener Verteilung der Inhalte (auch bei wechselnden Inhalten)
  - ... guter Darstellung auch bei Anpassung von Texten, Bildern, Farben (Anpassung von Sprachen, Kultur, ...)
- **Raster / Interaktion:** Alle Screen-Größen mit ...
  - ... guter Leitung der Aufmerksamkeit (Übersicht, Erwartungskonformität)
  - ... gut sichtbaren, begreifbaren (Affordance) und gut bedienbaren Controls
- **Struktur:** Alle Screen-Größen mit ...
  - ... einleuchtender, effizienter, findbarer Navigation (Dynamische Zwischenschritte für kleine Screens)
- **Umfang**
  - ... besser nicht abhängig von der Screen-Größe
- **Strategie**
  - Möglicherweise unterschiedliche Apps für unterschiedliche Aufgaben und Zielgruppen (nicht Screen-Größen)



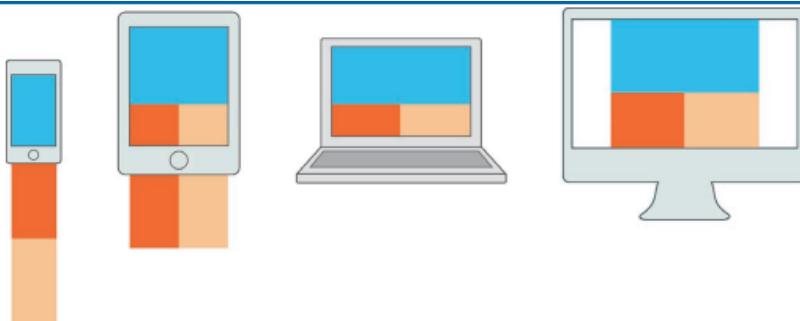
- **Oberfläche / Sensorisches Design:** Alle Screen-Größen mit ...
  - ... gut lesbaren Inhalten (Font, Zeilenlänge)
  - ... ausgewogener Verteilung der Inhalte (auch bei wechselnden Inhalten)
  - ... guter Darstellung auch bei Anpassung von Texten, Bildern, Farben (Anpassung von Sprachen, Kultur, ...)
- **Raster / Interaktion:** Alle Screen-Größen mit ...
  - ... guter Leitung der Aufmerksamkeit (Übersicht, Erwartungskonformität)
  - ... gut sichtbaren, begreifbaren (Affordance) und gut bedienbaren Controls
- **Struktur:** Alle Screen-Größen mit ...
  - ... einleuchtender, effizienter, findbarer Navigation (Dynamische Zwischenschritte für kleine Screens)
- **Umfang**
  - ... besser nicht abhängig von der Screen-Größe
- **Strategie**
  - Möglicherweise unterschiedliche Apps für unterschiedliche Aufgaben und Zielgruppen (nicht Screen-Größen)



- **Oberfläche / Sensorisches Design:** Alle Screen-Größen mit ...
  - ... gut lesbaren Inhalten (Font, Zeilenlänge)
  - ... ausgewogener Verteilung der Inhalte (auch bei wechselnden Inhalten)
  - ... guter Darstellung auch bei Anpassung von Texten, Bildern, Farben (Anpassung von Sprachen, Kultur, ...)
- **Raster / Interaktion:** Alle Screen-Größen mit ...
  - ... guter Leitung der Aufmerksamkeit (Übersicht, Erwartungskonformität)
  - ... gut sichtbaren, begreifbaren (Affordance) und gut bedienbaren Controls
- **Struktur:** Alle Screen-Größen mit ...
  - ... einleuchtender, effizienter, findbarer Navigation (Dynamische Zwischenschritte für kleine Screens)
- **Umfang**
  - ... besser nicht abhängig von der Screen-Größe
- **Strategie**
  - Möglicherweise unterschiedliche Apps für unterschiedliche Aufgaben und Zielgruppen (nicht Screen-Größen)

## Responsive Layout Patterns

Reflow nach dem Prinzip «Mostly Fluid»



## Web Engineering + Design 2

### Reflow nach dem Prinzip «Column Drop»



Abbildung 7.19 Modernizr als Beispiel für Column Drop

### Reflow nach dem Prinzip «Layout Shifter»

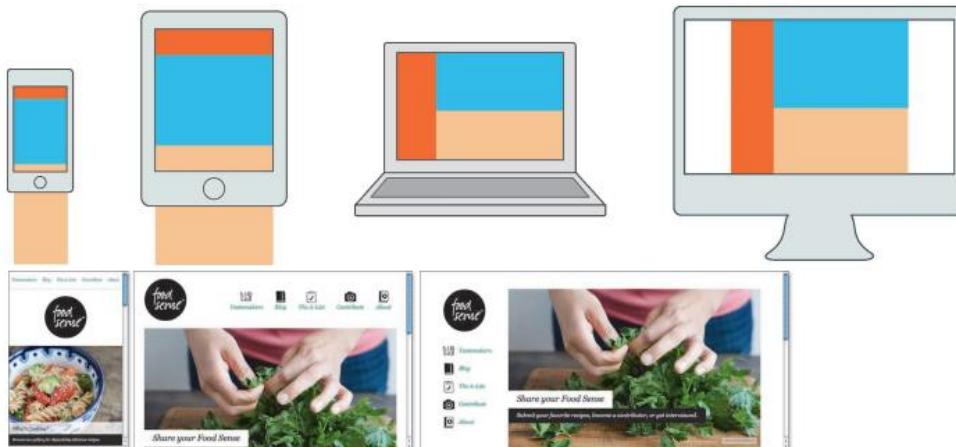


Abbildung 7.21 Bei Food Sense rutscht die Navigation in der großen Ansicht in die linke Spalte.

### «Reflow» Eine Spalte (Mobil) und viele Spalten (Desktop)

**Vorteile:** Der Platz wird sinnvoll genutzt. Es ist eine gute Grösse von Bilder vorhanden. Zudem ist die Zeilenlänge in den meisten Fällen gut.

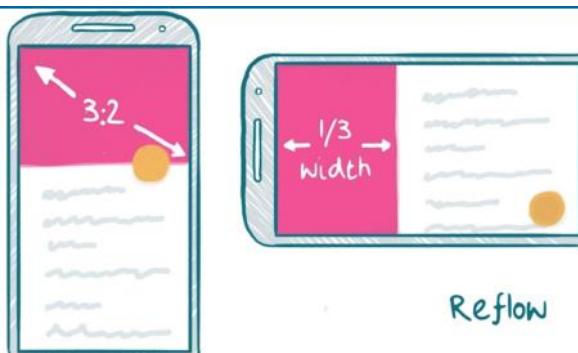


**Vorteile:** Gute Grösse von Bilder und eine gute Zeilenlänge.



Sidebar/SidePic für Landscape Ansicht

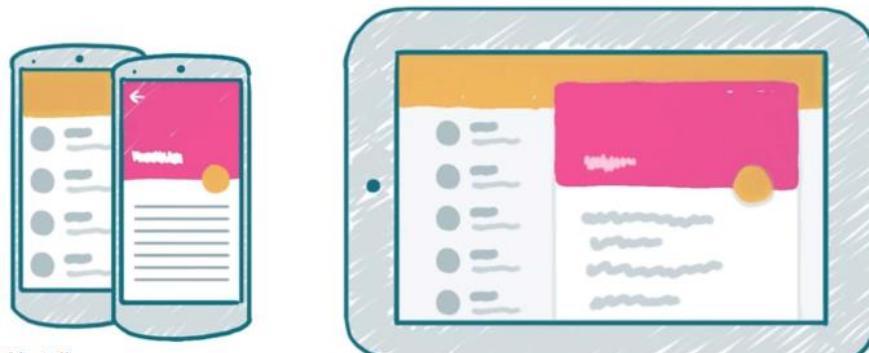
**Vorteile:** Hier wird die Bildschirmhöhe im Landscape Mode gut genutzt. Es ist genügend Platz für ein Bild vorhanden. Der Text hat eine gute Zeilenlänge.



Reflow

Master dann Detail(Mobile) und Master inklusive Detail (Tablet/Desktop)

**Vorteile:** Der Platz wird hier sinnvoll genutzt. Der Kontext bleibt erhalten. Zeilenlängen mit guter Lesbarkeit.



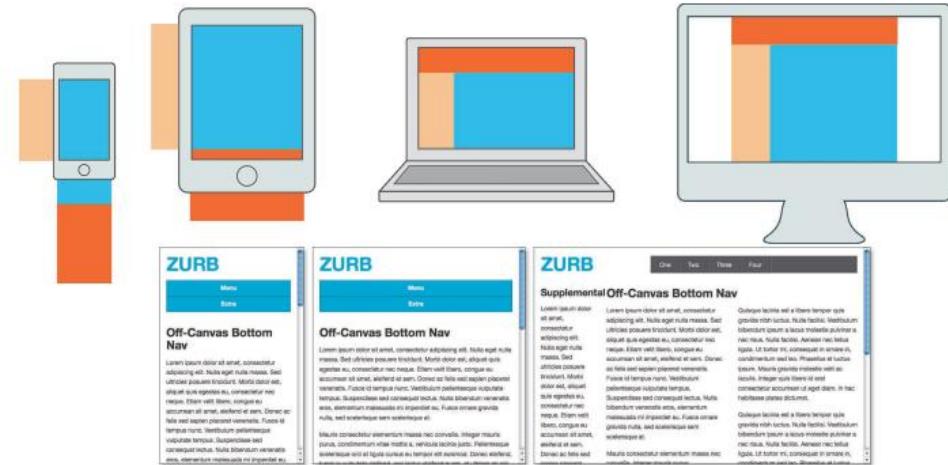


Abbildung 7.24 Layout der Footer-Navigation und Off-Canvas-Marginalie

### Off-Screen Menu(Mobile) und On-Screen Menu (Tablet/Desktop)

**Vorteile:** Platz sinnvoll genutzt, Kontext (aktueller Menü-Bereich) kann gezeigt werden.



### Layoutprobleme



Layoutprobleme, welche beim responsive Layout vermieden werden sollten.

- Unausgewogen (a, b, c)
- Schwierig zu lesen: zu lange oder zu kurze Zeilenlänge (a, b)
- Kontext ist (unnötig) nicht sichtbar (b)
- Bildqualität ist schlecht (b, c)

Progressive Disclosure

**Beispiel** Viel X-Platz → Tabs, Wenig X-Platz → Accordion

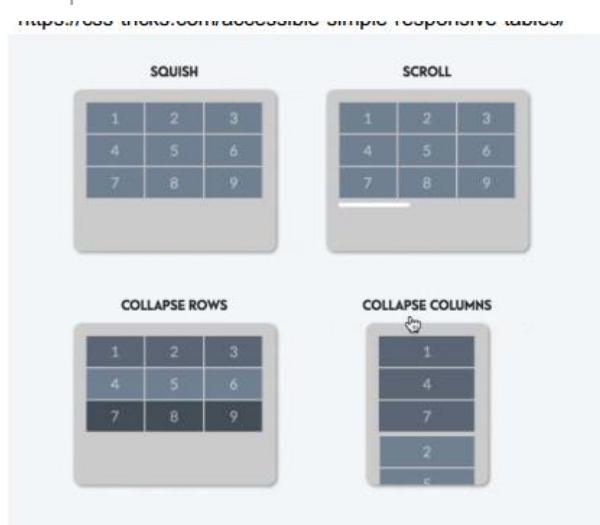
Die Umsetzung würde mit Flexbox stattfinden. Nachteil dabei ist der fehlende Überblick bei der Accordion, wenn erster Inhaltstext lang ist.

## Alternativen

- Link Liste (oder Teaser Text) mit Verweisen in die gleiche Seite (Scrolling Long Page)
- Gesamter Text in Spalten / Tabelle
- Menu Bar + Menu Button (Toggle Navigation)
- Kurze Tab Labels (oder mit ... gekürzt)
- Sliding Tabs (siehe CSS Tricks)
- (Carousell)



## Responsive Tables

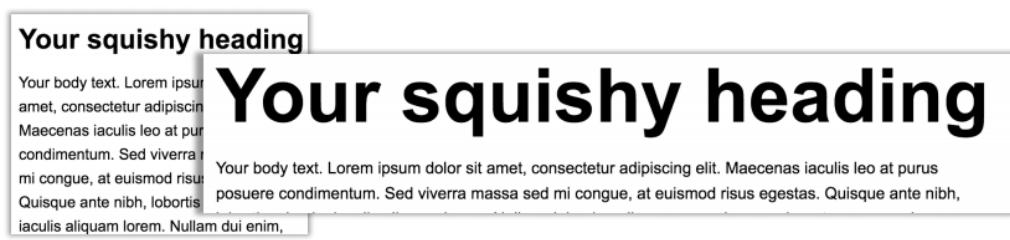


### Implementierbar mit Flexbox

- **Viel X/Y Platz →** Table wird ganz angezeigt
- **Wenig X-Platz**
  - o Squish (schmalere Spalten)
  - o Scroll (Scrollbar)
  - o Collapse Columns (Spalten am Ende anfügen)
- **Wenig Y-Platz**
  - o Collapse Rows (Zeilen entfernen)

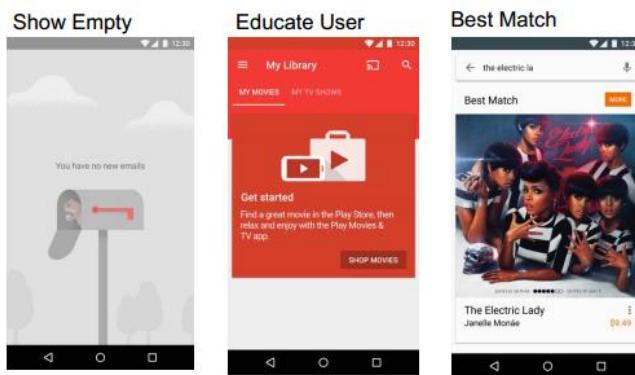
Headlines so gross wie möglich / nötig.

- |                      |                           |
|----------------------|---------------------------|
| <b>Viel X-Platz</b>  | Grosser Font für Headline |
| <b>Wenig X-Platz</b> | Kleine Font für Headline  |



Mehr Patterns (nicht nur Responsive)

Z.B. "Empty List / Empty State" Pattern <https://material.google.com/patterns/empty-states.html#>



(Responsive) Form Design

- |                     |                            |
|---------------------|----------------------------|
| <b>Viel X-Platz</b> | Label neben dem Input-Feld |
|---------------------|----------------------------|

- |                      |                           |
|----------------------|---------------------------|
| <b>Wenig X-Platz</b> | Label über dem Input-Feld |
|----------------------|---------------------------|

Einfach mit Flexbox



Label und Input in einem Container (z.B li) mit display:flex, flex-wrap: wrap und align-items: center platzieren.

Label mit Eigenschaften

flex: 1 0 120px; max-width: 220px;

Input mit Eigenschaften

flex: 1 0 220px

## To Bottom

YOUR NAME
Primary Phone
Message

<https://css-tricks.com/float-labels-css/>

## 2 PAYMENT Billing is same as shipping

Card Number

Month *	Year *	Security Code *
<input type="checkbox"/> Save this card for future purchases		
<b>CONTINUE TO REVIEW</b>		

<http://viget.com/inspire/making-infield-form-labels-suck-less>

«Review Mode» ohne Inputfelder spart Platz

## Aus G. Brzesinski, D. Triebswetter, U. v. Büren

Web Formular Design Generell

5. Message → Please review information that you have entered.

1. Label → First name\*  
2. Input Field → Justin  
Last name\*  
Mifsud

Your email address\*  
badEmail  
Repeat email\*

6. Validation → Email address not valid  
Note: no-one can see your email address.

By email  
saved  
charicat  
Can't read the text in the box?  
Refresh Listen Help

Type the text above here\*

Yes, I have read and I accept the Skype Terms of Use and the Skype Privacy Statement

I agree - Continue ← 3. Action

## 6 Elemente von Web Formularen

Labels

Input Fields

Help

Actions

Messages

Validation

## Ratschläge

Message/Validation beim Feld

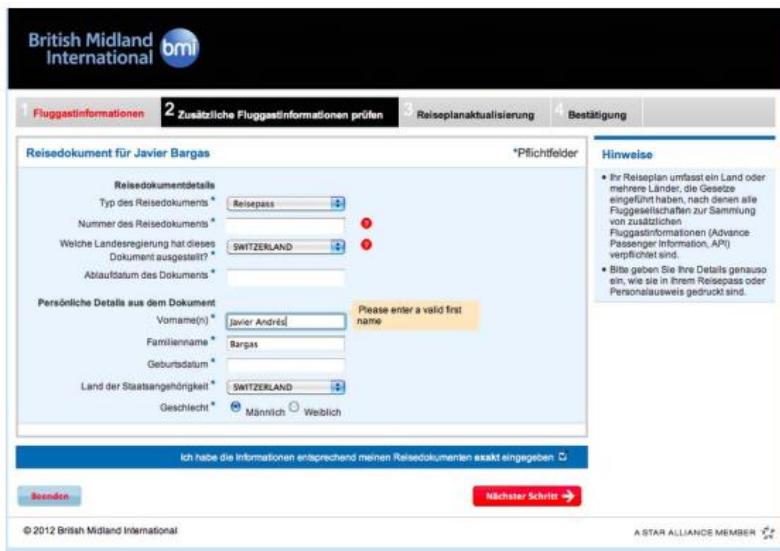
Validierung kontinuierlich

Keine «voreilige Validierung»

- Meldung nahe beim Fokus der Aufmerksamkeit des Nutzers zeigen
- Schon von der Validierung Beispiele von korrektem Input geben (z.B. 1.12.2016)
- Keine Beschuldigung (nicht «Sie haben die Daten falsch eingegeben»)
- Sprache des Nutzers sprechen (nicht «Fehler 27398» ohne weitere Erklärung)
- Hilfestellung bei der Behebung von Fehler geben («aktionsorientierte Hilfe»), z.B. «Postleitzahl und Ort müssen übereinstimmen» oder besser noch bei der Eingabe von PLZ den Ort ausfüllen und umgekehrt (aber nie ungefragt Benutzerinput überschreiben ohne das Undo möglich ist).

### Bad Practice

Validierung ohne Mass «My name is not valid»



The screenshot shows a step-by-step process for flight information entry:

- 1 Fluggastinformationen
- 2 Zusätzliche Fluggastinformationen prüfen
- 3 Reiseplanaktualisierung
- 4 Bestätigung

The current step is 2. The form contains the following fields:

- Reisedokument für Javier Bargas**
- \*Pflichtfelder**
- Hinweise**
  - Ihr Reiseplan umfasst ein Land oder mehrere Länder, die Gesetze eingeführt haben, nach denen alle Fluggesellschaften zur Sammlung von gesetzlichen Fluggastinformationen (Advance Passenger Information, API) verpflichtet sind.
  - Bitte geben Sie Ihre Details genauso ein, wie sie in Ihrem Reisepass oder Personalausweis gedruckt sind.

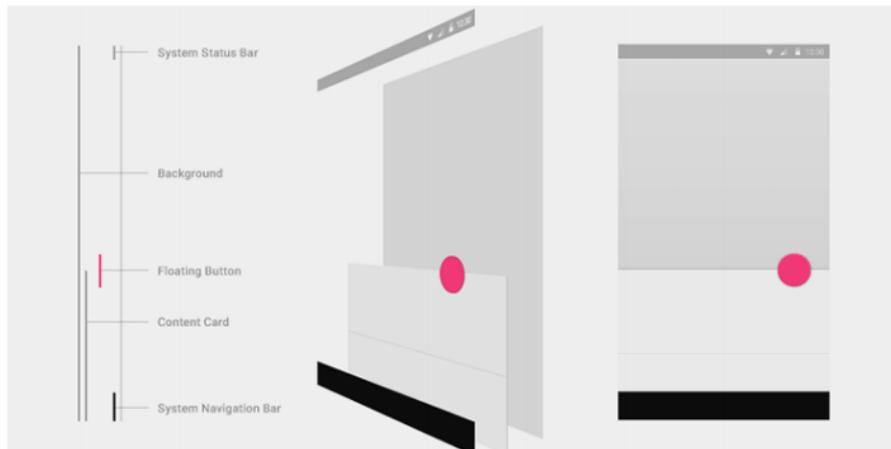
Personal details from the document:

Vorname(n) *	Javier Andrés	Please enter a valid first name
Familienname *	Bargas	
Geburtsdatum *		
Land der Staatsangehörigkeit *	SWITZERLAND	
Geschlecht *	<input checked="" type="radio"/> Männlich <input type="radio"/> Weiblich	

At the bottom, there is a note: Ich habe die Informationen entsprechend meinen Reisedokumenten exakt eingegeben.

Action buttons: **Senden** (Send) and **Nächster Schritt →** (Next Step →)

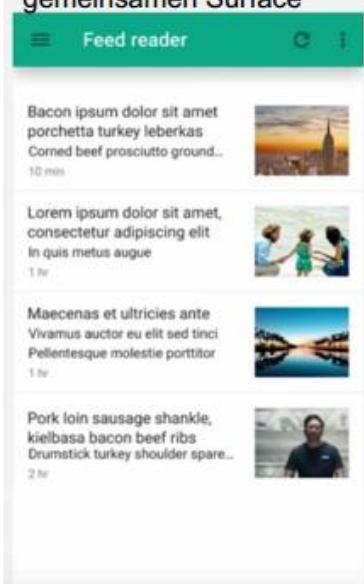
Material Design arbeitet mit übereinanderliegenden Papierflächen (surfaces) als Metapher. Der Fokus ist auf Mobil (Android), zunehmend findet es aber auch Anwendung im Web sowie auf iOS. Mehr dazu gibt es in der Vorlesung Mobil & GUO Engineering (MGE).



Schlecht: Zu viele Ränder



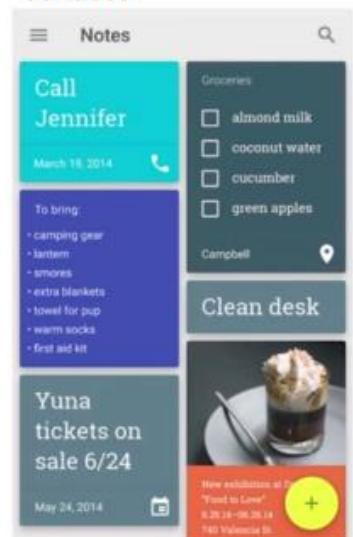
Besser: Alle Inhalte auf einer gemeinsamen Surface

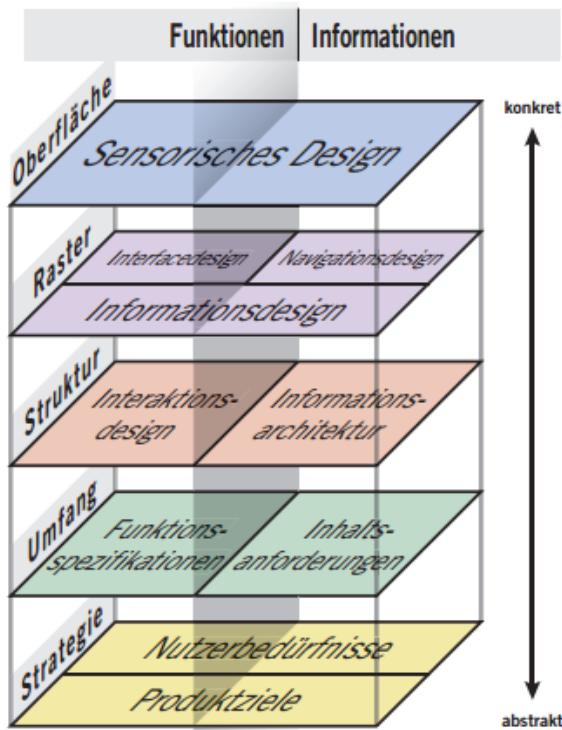


Schatten helfen Höhe von Surfaces zu kommunizieren. Design ist sonst aber eher nüchtern.

Nicht jeder Inhalt braucht eine eigene Surface. Zu viele Ränder (= Surfaces) stören den Lesefluss.

Besser: Unterschiedliche Inhalte auf (&lt; 5) separaten Surfaces





### ■ Oberfläche / Sensorisches Design

- Beachten: **Wahrnehmung** nur im Fokus
- Beachten: Eingeschränkte Wahrnehmung von Farben und kulturbedingte Interpretation

### ■ Raster-Design mit Interaktionsdesign

- Beachten: Hierarchie von Informationen, Leiten der Aufmerksamkeit des Leseflusses
- Optimieren: **Affordance von Controls** (Begreifbarkeit)
- Nutzen: Design Patterns

### ■ Strukturdesign

- Sicherstellen: User wissen immer:
  - Wo bin ich,
  - Was kann ich machen (wo kann ich hin),
  - Was ist passiert (wo komme ich her, wie zurück)?
- Sitemap, **Card-Sort**

## Sensorisches Design – Accessibility

Die Webseiten sollten auch für Farblinse verfügbar sein. Noch 2008 haben die Projektleiter diesem Thema zu wenig Beachtung geschenkt. Ab 2014 bis heute wird Accessibility als Werbethema eingesetzt. Bei Accessibility wird mehr als «nur» Wahrnehmung & Bedienung mit Screenreader getestet.



**Szenario:** Die Testperson möchte sich als Screenreader-Nutzerin einen Überblick über den Stundenplan einer beliebigen Vorlesung verschaffen und klärt ab, ob die entsprechenden Hörsäle rollstuhlgängig sind.

**Erfahrungsbericht Szenario:** Die PDF-Stundenpläne auf dieser Site konnten leider nicht gefunden werden. Ebenso konnten keine Informationen zur Rollstuhlgängigkeit der Hörsäle gefunden werden. Auffallend sind ausserdem übersprungene Überschriftenebenen in den Suchresultaten.

### Nicht-Test-Inhalte (Grafiken)

Informative Grafiken, einschliesslich Icons und verlinkte Grafiken, verfügen über sinnvolle Alternativtexte. Dekorationsgrafiken verfügen über leere Alt-Attribute. Es sind keine rein visuellen CAPTCHAs vorhanden.

### Tastaturbedienbarkeit

Alle interaktiven Elemente sind per Tastatur bedienbar. Alle Funktionalitäten können mittels Tastatur und von Screenreader-Anwendern bedient werden. Elemente mit Fokus, einschliesslich (unsichtbare) Sprunglinks, werden bei der Ansteuerung mit der Tastatur sichtbar hervorgehoben.

### Logische Reihenfolge

Die Reihenfolge der Links in der Navigation und im Inhalt ist logisch, auch bei ausgeschaltetem CSS. Inhalte in Tabellen werden richtig linearisiert.

## Semantische Struktur

Die Seiten der Website sind semantisch korrekt strukturiert, das heisst: Sie verfügen über eine Struktur, welche die inhaltlichen Bedeutungszusammenhänge korrekt abbildet. Die Struktur wird mittels Überschriften (h1-h6) und Landmarks, Listen, Headingzellen für Datentabellen und Labels für Formularfelder in HTML explizit ausgezeichnet. Auch Seitentitel und Sprachdeklarationen werden hier bewertet.

## Multimedia / 2-Sinne Prinzip

Für informative Multimedia-Inhalte (Audio und Video) existieren Textabschriften respektive Audiodeskriptionen, oder sie verfügen über synchrone Untertitel. Multimedia-Inhalte werden immer auch für mindestens einen alternativen Sinneskanal aufbereitet. Zusätzlich wird auch die Vermittlung von Information einzig durch Farbe unter diesem Punkt bewertet.

## Flexibilität der Anzeige

Die Art der Anzeige der Inhalte einer Seite kann von Menschen mit Einschränkungen entsprechend der spezifischen Bedürfnisse angepasst werden: Die Textgrösse lässt sich individuell anpassen. Sich bewegende Elemente können gestoppt werden. Zeitliche Limitierungen können vom Nutzer aufgehoben werden.

## Kontrast

Die Kontraste zwischen Hintergrund und Text ist ausreichend. Formularfelder sind gut sichtbar.

## Verständlichkeit

Überschriften, Formularbeschriftungen (Labels) und Linktexte sind selbsterklärend oder über den Kontext verständlich. Auf Formatwechsel (zum Beispiel PDFs) wird hingewiesen.

## Konsistenz / Vorhersehbarkeit

Die Navigation ist innerhalb eines Webauftrittes gleichbleibend angeordnet und aufgebaut. Elemente mit gleicher Funktion bleiben gleich. Kein Wechsel des Inhalts, wenn ein Element Fokus erhält. Änderungen des Inhalts bei Eingabe werden angekündigt.

## Syntax / Kompatibilität

HTML-Validierung mit Fokus auf Accessibility-relevante Aspekte. Korrektheit des Einsatzes von WAI-ARIA-Attributen. Kompatibilität mit verschiedenartigen und zukünftigen Ein- und Ausgeabegeräten.

## Hilfestellung bei Interaktionen

Feedback zu Formular-Validierungen hilft fehlerhafte Eingaben zu identifizieren. Pflichtfelder werden als solche ausgezeichnet. Eingaben mit rechtlichen Folgen können überprüft, geändert oder gelöscht werden.

## PDF Accessibility

PDFs werden mittels PDF Accessibility Checker auf PDF/UA Konformität und durch einen blinden Anwender auf alltagstaugliche Lesbarkeit geprüft.

### Tages-Anzeiger

[www.tagesanzeiger.ch](http://www.tagesanzeiger.ch)



**Szenario:** Die Testperson liest die tagesaktuellen News mithilfe des Screenreaders und lässt sich dabei von den eigenen Interessen leiten.



### Rangliste

<a href="http://www.letemps.ch">www.letemps.ch</a>	★★★★★
<a href="http://www.nzz.ch">www.nzz.ch</a>	★★★★
<a href="http://www.blick.ch">www.blick.ch</a>	★★★★
<a href="http://www.watson.ch">www.watson.ch</a>	★★★
<a href="http://www.laregione.ch">www.laregione.ch</a>	★★★
<a href="http://www.20min.ch">www.20min.ch</a>	★★★
<a href="http://www.24heures.ch">www.24heures.ch</a>	★★
<a href="http://www.bazonline.ch">www.bazonline.ch</a>	★
<a href="http://www.bernerzeitung.ch">www.bernerzeitung.ch</a>	★
<a href="http://www.luzernerzeitung.ch">www.luzernerzeitung.ch</a>	★
<a href="http://www.cdt.ch">www.cdt.ch</a>	★
<a href="http://www.tagesanzeiger.ch">www.tagesanzeiger.ch</a>	★
<a href="http://www.derbund.ch">www.derbund.ch</a>	★
<a href="http://www.lematin.ch">www.lematin.ch</a>	★
<a href="http://www.tdg.ch">www.tdg.ch</a>	★

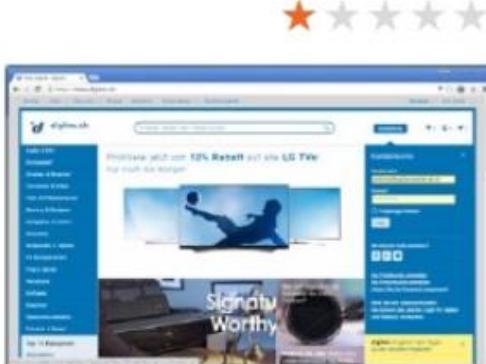
### Digitec

[www.digitec.ch](http://www.digitec.ch)



**Szenario:** Die Testperson möchte eine Bestellung verschiedener Artikel vornehmen. Dazu muss sie sich registrieren, die gewünschten Artikel auswählen und den Warenkorb überprüfen.

**Erfahrungsbericht Szenario:** Die Überschriftenstruktur von Tages-Anzeiger.ch ist mangelhaft. Überschriftenebenen werden übersprungen. Grafische Inhalte sind weitgehend unbeschrikt. Störend sind die Links zu sozialen Medien zwischen Artikel-Überschriften



**Erfahrungsbericht Szenario:** Die Registrierung mithilfe eines Screenreaders scheitert an nicht korrekt mit Beschriftungen (Labels) verknüpften Formularfeldern. Der Suchen-Button ist nicht beschriftet. Popups mit spezifischen Informationen zu Warenkorb, Kasse etc. werden am Seitenende angezeigt und sind somit nur schwer zu finden. Bestellungen bei Digitec sind für SR-Nutzer äusserst fehleranfällig.

### Rangliste

<a href="http://www.brack.ch">www.brack.ch</a>	★★★★★
<a href="http://www.manor.ch">www.manor.ch</a>	★★★★
<a href="http://www.books.ch">www.books.ch</a>	★★★★
<a href="http://www.zalando.ch">www.zalando.ch</a>	★★★★
<a href="http://www.weltbild.ch">www.weltbild.ch</a>	★★★★
<a href="http://speedyshop.ch">speedyshop.ch</a>	★★★
<a href="http://www.coopathome.ch">www.coopathome.ch</a>	★★★
<a href="http://www.conrad.ch">www.conrad.ch</a>	★★★
<a href="http://www.leshop.ch">www.leshop.ch</a>	★★★
<a href="http://www.exlibris.ch">www.exlibris.ch</a>	★★★
<a href="http://www.digitec.ch">www.digitec.ch</a>	★

## Accessibility Regeln – Technologieunabhängig

- Jedem Bild und jeder Animation ist eine angemessene textuelle Beschreibung direkt zugeordnet.
- Es wird keine Information ausschliesslich durch Farbe dargestellt
- Vorder- und Hintergrund sich auch bei reduzierter Farb- und Kontrastwahrnehmung in der Standardansicht deutlich unterscheidbar.
- Eine Skalierung der Schrift über Funktionen des Browsers ist möglich.
- Jegliche Funktion der Seite ist auch über die alleinige Verwendung der Tastatur in einer schlüssigen Reihenfolge zu erreichen.
- Beschwerden und Nachfragen müssen auch schriftlich gestellt werden können.

## Wichtigste Regeln um HTML Accessibility zu verbessern

- Wichtigste Seitenelemente am Start der Seite (DOM Baum)
  - o Navigation zuerst

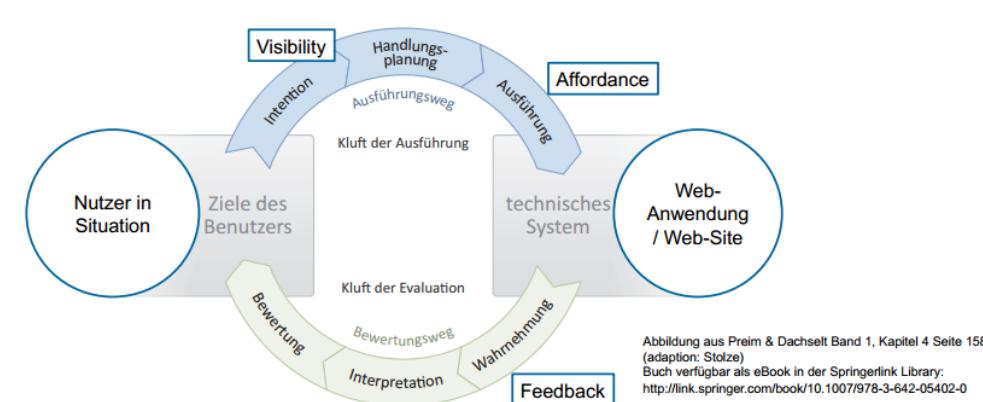
## Web Engineering + Design 2

- Bereiche gruppieren
- Alt == Beschreibungen
- Semantic Markup nutzen
  - Nav, header, main, h1
  - NICHT div, span (z.B. gestyled als Button)
    - Statt <span onclick=>>> lieber <a href=>>>
    - Statt <div onclick=>>> lieber <button onclick=>>>
  - Nicht Tabellen im Layout
  - Tabellen für Daten mit
    - Column Heading: thead (table Header) mit column Headings <th scope>..</th>
    - Row Heading <tr> mit <th scope="row">...</th>
    - Caption zur Beschreibung
  - Forms mit <label> Elementen für <input> Elemente
    - HTML 5 Validation nutzen

## Interaktionsdesign – Design von Controls

Die wichtigsten Problemgruppen bei der Nutzung von Interaktiven Systemen sind Visibility, Affordance und Feedback.

Die Theorie zur Interaktion mit Systemen (Don Norman, Design of Everyday Things) postuliert: Bei der Nutzung von Systemen wechselt sich Handeln (Ausführung) und Beobachten/Bewerten (Evaluation) ab. Fehlende Sichtbarkeit (Visibility), Affordance (Begreifbarkeit) und Feedback verhindern, dass die notwendigen Schritte der Interaktion sinnvoll durchlaufen werden können.



Die Affordance hängt ab von der visuellen Gestaltung der Controls und der Erfahrung/Erwartung von Nutzern.



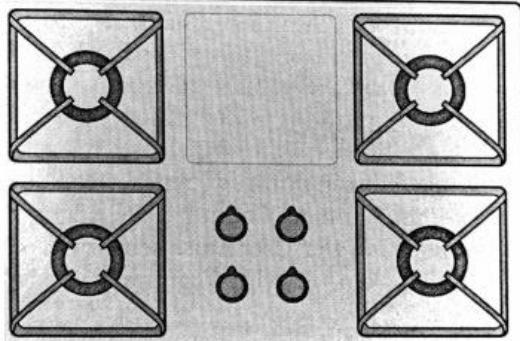
## Affordances von Controls – Was wird verstanden

- Links (Navigation) Blau und unterstrichen
- Logo oben links (oder rechts) führt zurück auf die Homepage

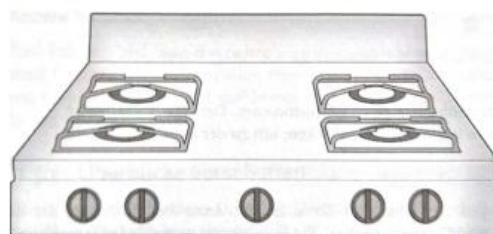
- Grauer Hintergrund-Text ist ein Label oder eine Instruktion
- Login/Nutzerinformation ist oben rechts
- Navigation-Menü ist oben (manchmal links oder rechts mit Hamburger-Button)
- Suchfelder haben ein Lupe-Icon

### Beispiele von Affordance

Gute Affordance durch klares Mapping



Schlechte Affordance durch unklares Mapping



D. Norman (1988) The Design of Everyday Things

### Schlechtes Feedback (und schlechte Affordance)

Nicht klar ob Ziel erreicht

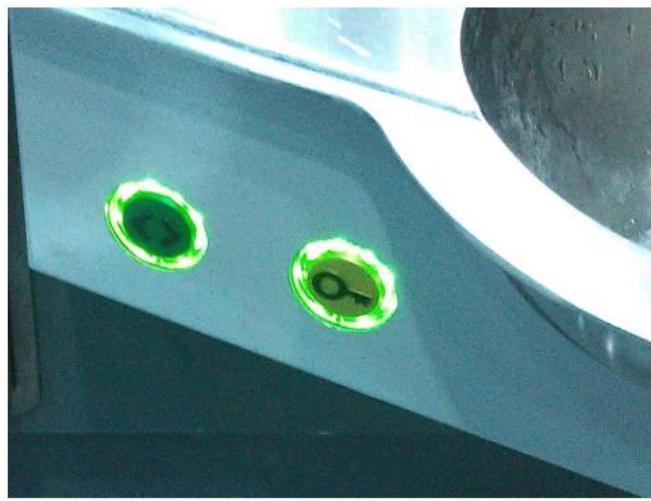


Bild Tobias Blaser 2015, MOB, Golden Pass Express

### Struktur Design – Card Sort etc.

Design der Site-Struktur: Information «Scent» mittels «Card Sorting» optimieren. Herausforderung bei der Navigation ist der Information Scent. Links sollten so benannt sein, dass klar ist welche Ziele durch die Navigation erreicht (und nicht erreicht) werden.

### ■ Schritt 1: Content Repository Erstellen (Was sind Zielpunkte der Navigation)

#### ■ Schritt 2: "Open Card Sort"

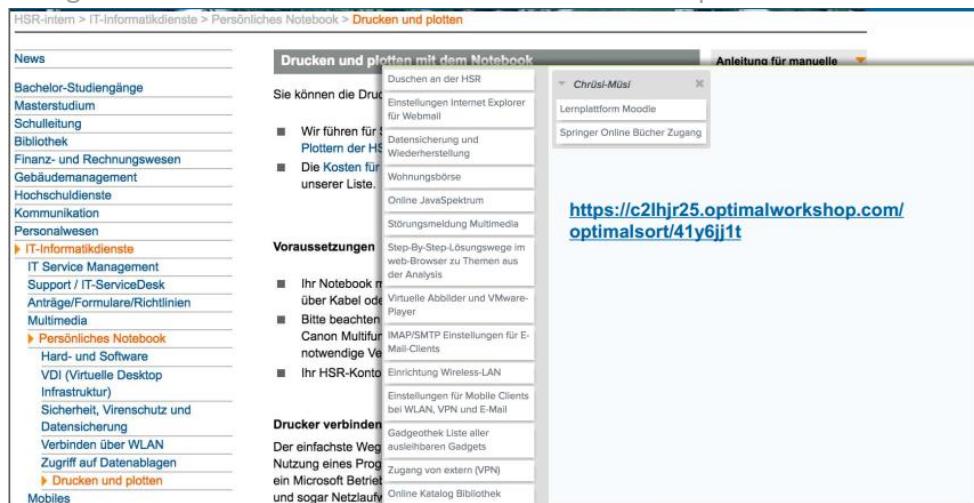
- 2.1) 5+ Personen aus der Zielgruppe rekrutieren
- 2.2) Content Elemente in disjunkte Gruppen ordnen lassen
- 2.3) Gruppen benennen lassen

#### ■ Schritt 3: Gute Gruppennamen identifizieren (Hypothese)

#### ■ Schritt 4: Gruppennamen mittels "Closed Card Sort" Validieren

- 4.1) 5+ Neue Personen aus der Zielgruppe rekrutieren
- 4.2) Gruppennamen vorgeben
- 4.3) Content Elemente den Gruppen zuordnen lassen

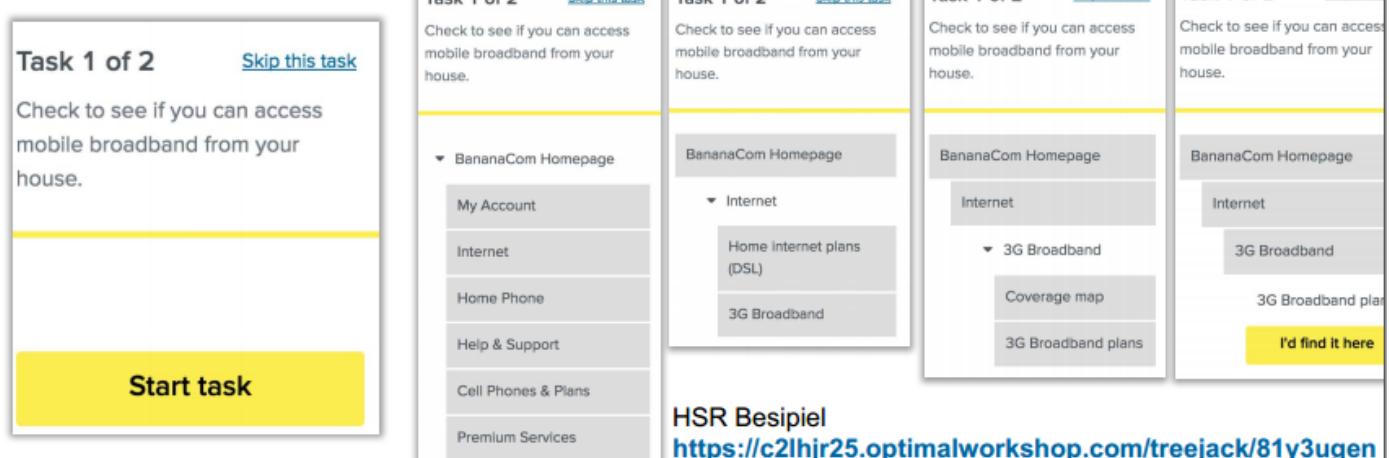
Navigationsstruktur mit Cardsort bestimmen am Beispiel des HSR Intranet mit OptimalSort



Alternative bei bestehenden Sites – Tree Testing

Die aktuelle Baumstruktur (Site Map) aufnehmen. Aufgaben (Szenarios) zur Erreichung von Zielen stellt. Testen.

#### ■ Beispiel (Probandensicht)



**Task 1 of 2** [Skip this task](#)

Check to see if you can access mobile broadband from your house.

**Start task**

**Task 1 of 2** [Skip this task](#)

Check to see if you can access mobile broadband from your house.

BananaCom Homepage

- My Account
- Internet
- Home Phone
- Help & Support
- Cell Phones & Plans
- Premium Services

**Task 1 of 2** [Skip this task](#)

Check to see if you can access mobile broadband from your house.

BananaCom Homepage

- Internet
- Home Internet plans (DSL)
- 3G Broadband
- Coverage map
- 3G Broadband plans

**Task 1 of 2** [Skip this task](#)

Check to see if you can access mobile broadband from your house.

BananaCom Homepage

- Internet
- 3G Broadband
- 3G Broadband plans

**Task 1 of 2** [Skip this task](#)

Check to see if you can access mobile broadband from your house.

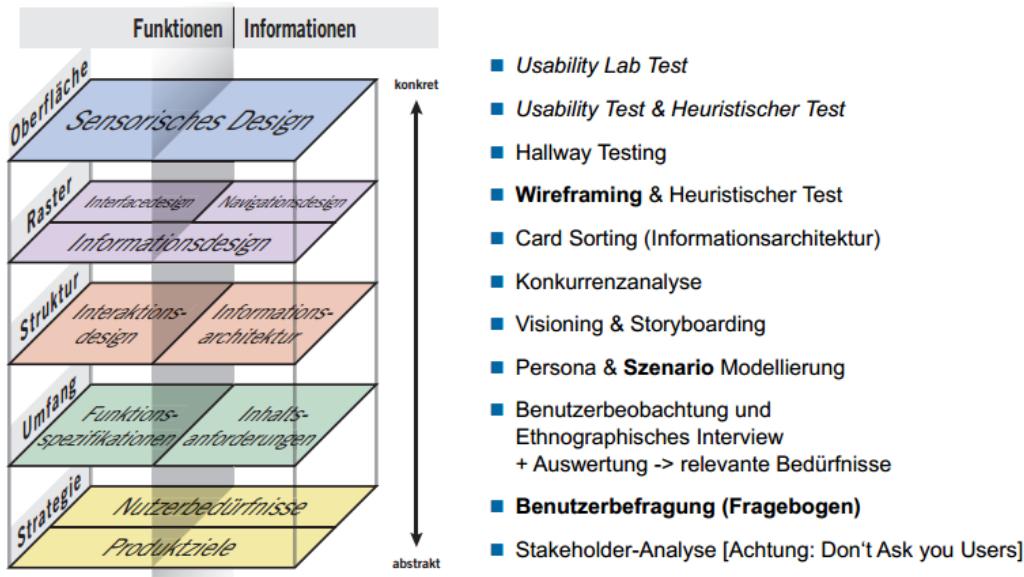
BananaCom Homepage

- Internet
- 3G Broadband

**I'd find it here**

**HSR Beispiel**  
<https://c2lhjr25.optimalworkshop.com/treejack/81y3ugen>

## Wichtigste Techniken des User Centered Design sortiert nach Garrett Ebenen

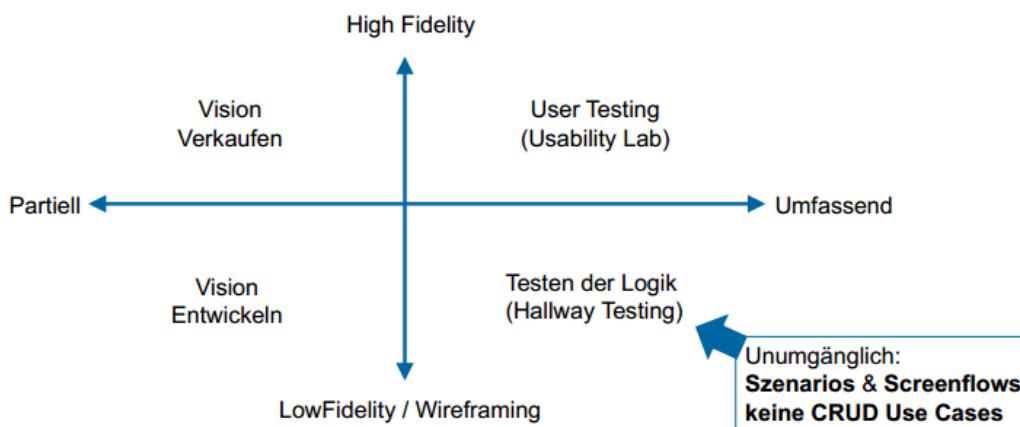


### Scenarios & Wire-framing / Testing

Prototypen unterscheiden sich in zwei Dimensionen

- Ihrer visuellen & interaktiven Exatheit: High Didelity vs. Low Fidelity
- Ihrer Vollständigkeit: Umfassend vs. Partiell

Es ergeben sich 4 unterschiedliche Typen von Prototypen für unterschiedliche Ziele.



Die Grösste Herausforderung beim Usability Test ist die Erstellung guter Aufgaben. Gute Aufgabenstellungen machen echte Benutzerziele zur Aufgabe (nicht «sichern Sie ihr File»). Verraten Sie keine «Keywords».

■ **Melden Sie sich bei xxxx mit folgenden Daten an:**

- Benutzername: Peter Muster
- Passwort: 1234567

■ **Definieren sie eine neue Region**

- Stadt St. Gallen
- Passendes Gebiet

So nicht!

■ **Stellen Sie folgende Frage**

- Überschrift: Beste Pizza in St. Gallen
- Frage: [passend zu Titel selbst definieren]

■ **Geben Sie eine Antwort zur Frage „Indie Konzert“  
(ohne eine Lokation anzuhängen)**

**Gute Aufgaben**

■ **Sie sind mit einigen Mitgliedern einer Indie Band CoolKids befreundet.  
Um diese Band promoten zu können haben Sie sich die Android  
Applikation LocalHeros heruntergeladen und sich registriert.  
Dabei verwendeten Sie folgende Daten:**

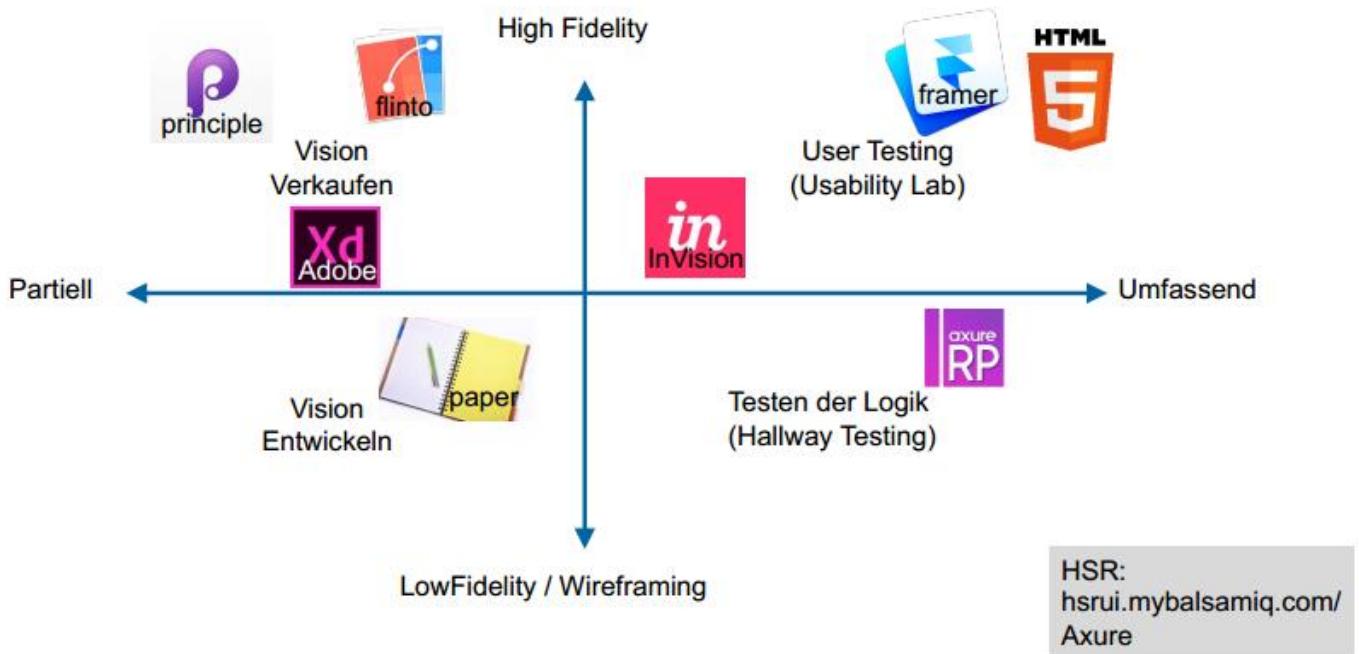
- Benutzername: Peter Muster
- Passwort: 1234567

■ **Sie meinen, dass bei LocalHeros Anfragen von Interessenten  
Ausgehtipps in St. Gallen angezeigt werden sollten. Nach einer Woche  
erhalten sie noch immer keine Meldungen zu neuen Fragen. Finden Sie  
heraus, woran das liegt und versuchen Sie, dies zu ändern.  
Promoten dann Ihre Band.**

■ **Sie befinden sich gerade mit ihrer Freundin,  
welche Hunger auf Pizza hat, in der Stadt St. Gallen.  
Verwenden Sie LocalHeros, um die beste Pizza der Stadt zu finden.**

**Prototyping Tools**

	Interaction	Animation & Motion	Gesture Support	Visual Design	Simplicity in Demoing	Digital Collaboration	Documentation	Responsive Design
Good Fit	AppCooker Blueprint InVision Keynote Marvel POP	AppCooker Axure Blueprint Flinto InVision Justinmind Marvel POP Proto.io	AppCooker Blueprint Flinto InVision Marvel POP Proto.io	Flinto FramerJS InVision jQuery Mobile Marvel Pixate POP	Axure CSS3 InVision jQuery Mobile Marvel Proto.io	AppCooker Blueprint Flinto FramerJS Keynote Marvel Pixate POP Proto.io InVision	AppCooker Blueprint Flinto Pixate POP Proto.io Keynote Marvel	AppCooker Blueprint Flinto FramerJS Keynote Marvel Pixate POP InVision
More Suitable	Axure CSS3 Bootstrap Flinto FramerJS jQuery Mobile Justinmind Pixate Proto.io	CSS3 FramerJS Keynote jQuery Mobile	Axure FramerJS HammerJS Justinmind Pixate	AppCooker Axure BluePrint CSS3 Justinmind Proto.io	AppCooker Blueprint Flinto FramerJS Justinmind Marvel Pixate POP	Axure CSS3 Justinmind jQuery Mobile	Axure CSS3 FramerJS jQuery Mobile Justinmind	Axure Bootstrap CSS3 Foundation HotGloo Justinmind Proto.io WebFlow



### Fragebögen

Fragebögen sind einfach und ergiebig. Einfach zu erstellen mit Google und zur verschicken. Resultat sind Daten, welche Faken sind.

### Fragebögen sind schwierig

- Sollten getestet sein
- Sollten an eine repräsentative Gruppe geschickt werden
- Sollte von einer repräsentativen Gruppe beantwortet werden. Viele Antworten heisst nicht unbedingt repräsentativ.
- Resultate benötigen Interpretation
- Resultate benötigen einen Vergleichspunkt (Base-Line)

### Usability Baseline erfassen vor einer UX Massnahme (und Verbesserung messen)

- z.B. SUS Fragebogen  
<https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>  
<https://minds.coremedia.com/2013/09/18/sus-scale-an-improved-german-translation-questionnaire/>
- ISO Norm Fragebogen  
[http://www.ergo-online.de/site.aspx?url=html/software/verfahren\\_zur\\_beurteilung\\_der/fragebogen\\_isonorm\\_online.htm](http://www.ergo-online.de/site.aspx?url=html/software/verfahren_zur_beurteilung_der/fragebogen_isonorm_online.htm)

### Herausforderung: Umfrage repräsentativ für Fragestellung machen

- Sollen auch nicht-Kunden eingeschlossen werden
- Wie verhindert man "self selection" (oder wie geht man damit um)

⇒ Schlussfolgerung: Traue keiner Statistik...

## Intro

Web Security ist wichtig. Zu den Top Web Applications «Fails» gehört, dass sie nicht sicher sind. Dies führt zu Imageverslust und hat ev. auch rechtliche Folgen bzw. Haftung. Ein späterer Einbau von Sicherheit ist aufwändig und unsicher.

### Mini Fallstudie – Yahoo XSS Vulnerability

#### Verwundbarkeit

Die HTML Darstellung des Yahoo E-Mail Clients war zu wenig gesichert: Speziell präparierte HTML E-Mails konnten enthaltenes JS im Kontext des Yahoo Web Mail-Readers beim Anschauen zur Ausführung bringen.

#### Angriff

Da Yahoo Nutzer beim Lesen von E-Mail in ihren Account eingeloggt sind, wäre es möglich gewesen den gesamten Inhalt der Inbox an einen WebSite eines Angreifers zu senden, oder auch die Inbox vollständig zu löschen.

#### Gegenmassnahme

##### Verbesserung des Encodings / Filtering

##### Web Security = OWASP Top 10

1. **Injection: JS Code Injection, SQL Injection, ... (VL + Ü: 2)**
2. **Broken Authentication and Session Management (VL + Ü: 3)**
3. **Cross-Site Scripting - XSS (VL + Ü: 1)**
4. Insecure Direct Object References (VL: 4)
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (VL: 5)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

### Cross-Site-Scripting (XSS)

#### Definition

Eine Web-Site besitzt eine XSS Verwundbarkeit, wenn es möglich ist den Server (bzw. die Daten des Servers) so zu manipulieren, dass Schadcode (Javascript) eines Angreifers an Nutzer (Opfer) ausgeliefert wird und im Browser dieser Nutzer ausgeführt wird.

#### Demobeispiel mit Node (Note-Goat)

#### Verwundbarkeit

Von Nutzern in einem Formular eingegebene Daten werden ohne «escaping» oder anderwertige Nachbearbeitung an andere Nutzer ausgeliefert. z.B. User-Name eines Nutzers wird in der Admin Konsole dem Administrator angezeigt.

#### Angriff

Der Angreifer gibt im Input Feld statt regulären Text HTML ein. Z.B. <source>function read.....</source>

#### Gegenmassnahme

Kontext-sensitives Data Encoding bei der Darstellung (Output) von Nutzer-Input Daten. Frameworks wie

Handlebars haben encoding per Default aktiviert. Achtung daher besonders bei Abschnitten die aus Sicht Entwickler "extra" unescaped ausgeliefert werden (z.B. um <pre> oder <em> Tags im Text zu erlauben). Für manuelles Kontext-sensitives Data Encoding kann die folgende Tabelle (siehe Details-1 Folie) genutzt werden

### Übersicht Kontext-sensitives Encoding gegen XSS

Context	Code Sample	Encoding Type
HTML Entity	<span>UNTRUSTED DATA</span>	Convert & to &amp; Convert < to &lt; Convert > to &gt; Convert " to &quot; Convert ' to &#x27; Convert / to &#x2F;
HTML Attribute Encoding	<input type="text" name="fname" value="UNTRUSTED DATA">	Except for alphanumeric characters, escape all characters with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value)
URI Encoding	<a href="/site/search?value=UNTRUSTED DATA">clickme</a>	Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value)
JavaScript Encoding	<script>var currentValue='UNTRUSTED DATA';</script><script>someFunction('UNTRUSTED DATA');</script>	Ensure JavaScript variables are quoted. Except for alphanumeric characters, escape all characters with ASCII values less than 256 with \uXXXX unicode escaping format (X = Integer), or in xHH (HH = HEX Value) encoding format.
CSS Encoding	<div style="width:UNTRUSTED DATA;">Selection</div>	Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the \HH (HH= Hex Value) escaping format.

### Weitere Sicherungsmassnahmen gegen XSS

#### Input Validierung (oder Reinigung)

bei der Entgegennahme von Daten

#### Content Security Policy (CSP) im Header setzen

z.B. nur die Source Domain und eine REST Service darf kontaktiert werden: *Content-Security-Policy: default-src 'self' \*.myRestServiceDomain.com*. Externe und Inline-Scripts sind hiermit verboten. Dies kann (aber sollte nicht) mit dem Zusatz 'unsafe-line' wieder erlaubt werden.

*Content-Security-Policy: default-src 'self' 'unsafe-line' \*.myRestServiceDomain.com*

#### Bei Cookies soweit möglich das HTTPOnly Flag setzen

z.B. gesichertes Session Cookie bei Express

```
app.use(express.session({
  secret: "server secret",
  key: "sessionId",
  cookie: {
    httpOnly: true,
    secure: true
  }}))
```

Aug 4, 2015 07:13 GMT · By Catalin Cimpanu  · Share:    

**Popcorn Time, the BitTorrent-based software that allows users to stream and watch movies, is vulnerable to XSS, local file reading, and remote code execution attacks, as Antonios A. Chariton, a security engineer and researcher, has discovered.**

### The Node.js codebase escalated the whole problem

Since the whole client is coded in JavaScript (Node.js) and has access to the user's entire PC, attackers could easily gain control of a computer, obtaining local files and executing their own malicious code inside the application just by packing it inside an API response.

All they need is the JavaScript know-how to get this done. Skills Mr. Chariton had, proving in his experiments that he could easily read and write local files.

<https://blog.daknob.net/grab-some-popcorn-and-launch-popcorn-time/>

```
"data": {  
    "movie_count": 4308,  
    "limit": 50,  
    "page_number": 1,  
    "movies": [{  
        "id": 4332,  
        "imdb_code": "tt2967224",  
        "title": "Hot Pursuit",  
        "title_english": "Hot Pursuit",  
    }]
```

JS Injection:  
API Server liefert JSON mit <script> Tags (XSS)

Z.B. Titel erweitert mit script tag:

Hot Pursuit <script>alert('XSS');</script>

Oder  
Json = ...  
"title": (alert("Injected"), "Hot Purs...  
...  
Und Parsing mit eval("(+json+)")

### Definition

Ein Web-Server besitzt eine Code Injection "Vulnerability" wenn ein Angreifer den Server dazu bringen kann vom Angreifer eingeschleusten Code zum Ausführen bringen. Bei Node Servern ist dies entsprechend JavaScript, welches zur Ausführung untergeschoben wird.

### Demo von NodeGoat

JS Code Injection: Demo von NodeGoat: <http://nodegoat.herokuapp.com/tutorial/a1>

### Verwundbarkeit

Eval wird im Post-Handler des «Contributions» Formular genutzt um einen (vermeintlichen) nummerischen Wert zu konvertieren.

```
app.post("/contributions", isLoggedIn, contributionsHandler.handleContributionsUpdate);  
[...]  
function ContributionsHandler(db) {  
[...]  
    this.handleContributionsUpdate = function(req, res, next) {  
        // Insecure use of eval() to parse inputs  
        var preTax = eval(req.body.preTax);  
        var afterTax = eval(req.body.afterTax);  
        [...]}
```

### Angriff: Eingabe von JS im Feld preTax z.B.

- while(true) (Endlosschleife) oder process.exit() (Exit) -> Denial-of-Service-Attack
- res.end(require('fs').readdirSync('.').toString()) (Directory lesen, dann File lesen, ....)

### Gegenmassnahmen

- NICHT eval() (oder setTimeOut(), setInterval()) nutzen sondern parseInt(), JSON.parse()
- Im Minimum Input säubern und/oder mittels Regex auf korrektes Format testen

### **Globale Scopes und Variablen reduzieren**

Dies macht es aufwändiger aus einem Controller den ganzen Server zu übernehmen.

### **Rechtenintensive Tasks mit childprocess.spawn auslagen, um DDOS Attacken zu erschweren**

#### **Node Nicht als root Prozess starten**

Bzw. root Privilegien nach Start auf Port 80 abgeben

#### **Die x-powered-by Header Informationen abstellen**

app.disable('x-powered-by'); und statt der express-spezifischen, generischen Cookie-Namen definieren.

Weiter zu beachten sind auch SQL Injection oder NoSQL Injection.

Broken Authentication and Session Management

#### **Definition**

Bereiche eines Web-Sites mit benutzer-spezifischen Informationen sollten nur für authentisierte Nutzer zugreifbar sein. Bei Problemen bei der Authentisierung und dem Session Management können externe Angreifer oder Angreifer mit einem validen Login auf Informationen zugreifen, welche nicht für Sie bestimmt sind.

#### **Beispiel 1**

#### **Verwundbarkeit**

Passwort oder Token wird nicht verschlüsselt übertragen. Unabhängig davon ob als URL Query-Parameter (get)oder im Request-Body(post).

#### **Angriff**

Der Angreifer nutzt Network Sniffing Software z.B. in einem offenen WLAN und extrahiert die Informationen.

#### **Gegenmassnahme**

https(TLS) nutzen

#### **Beispiel 2**

#### **Verwundbarkeit**

Der Session Timeout ist zu lang

#### **Angriff**

Wenn Nutzer einen öffentlichen PC nutzt und sich nicht auslogged, kann ein Angreifer den Browser erneut starten und ist authentisiert.

#### **Gegenmassnahme**

Session Timeout sinnvoll setzen (mindestens für sensible Daten und Operationen)

#### **Beispiel 3**

#### **Verwundbarkeit**

Passwörter und User unverschlüsselt in Server DB gespeichert.

#### **Gegenmassnahme**

Verschlüsseln

## Definition

Eine Web-Site besitzt eine Insecure Direct Object Reference Verwundbarkeit, wenn es möglich ist über einen manipulierten Datenschlüssel auf Daten zuzugreifen für die Nutzer die der Nutzer keine Berechtigung hat.

Beispiel mit NodeGoat

<http://nodegoat.herokuapp.com/tutorial/a4>

## Verwundbarkeit

Die RetriEasy Anwendung ist verwundbar da die User-ID für die Anzeige des Aktienportfolios nur die URL codiert und nicht separat sicher gestellt wird, dass der aktuell eingeloggte Nutzer Zugriffsrechte hat.

## Angriff

Der Angreifer loggt sich ein und manipuliert die URL der StockPortfolio Seite von ..../:attackerUID Zu ..../:victimUID. Diese Seite wird dann angezeigt.

## Gegenmassnahme

Bei allen Seiten mit benutzerspezifischen Daten sollte vor der Anzeige sicher gestellt werden, dass der aktuell eingeloggte Nutzer berechtigt ist. Wenn Express genutzt wird, lässt sich dies durch die Spezifikation einer zusätzlichen Middleware-Funktion in den Route-Definitionen erreichen. Diese Middlewarefunktion überprüft ob eine aktuelle UserID in der Request Session gesetzt ist. Dieses Session Objekt wird von Express angelegt. Beim erfolgreichen Login speichert der entsprechende Login-Handler die User-ID im Session Objekt.

- Post-Handler für die das Login Formular: Setzt bei Erfolg req.session.user\_id
  - app.post('/login', function (req, res) {  
 var user = findUser(req.body.name);  
 if( !user && user.checkPassord(req.body.password))  
 {  
 req.session.user\_id = user.id;  
 res.redirect('/home/index.html');  
 }  
 res.redirect('/login');  
});
- Middleware Funktion welche sicher stellt, dass Nutzer eingelogged sind
  - function requireLogin(req, res, next) {  
 if (typeof(req.session.user\_id) == "number") {  
 next();  
 } else {  
 res.send(401, 'not authorized');  
 }  
}
- Middleware Route Definition für den geschützten Bereich
  - app.all("/api/\*", requireLogin, function(req, res, next) {  
 next(); // if the middleware allowed us to get here, just move on  
});

## Cross Site Request Forgery

### Definiton / Therorie

Eine Webseite besitzt eine Cross Site Request Forgery Verwundbarkeit wenn es möglich ist das bei diesem Site vom Nutzer unbemerkt und unter Nutzung einer noch nicht ausgelaufenen Session sicherheitsrelevante Operationen ausgeführt werden können.

## Verwundbarkeit

Die RetireEasy Anwendung ist verwundbar da das «Profile» Formular nicht sicher stellt, dass das Formular vom Nutzer selber ausgefüllt wurde.

## Angriff

Der Nutzer ist aktuell bei RetireEasy eingelogged, er wird aber (z.B. über einem Link in einem E-Mail) auf eine Seite mit Schadcode gelockt. Diese Seite generiert einen Formular-Post-Request auf die RetireEasy/profile Seite mit gefälschten Daten für das Profil.

## Gegenmassnahme

Um zu verhindern, dass Requests gefälscht werden, sollten server-seitig Security Tokens generiert werden. Diese werden jeweils in das Nutzern angezeigte Formular eingebettet. Mit einem vom Nutzer korrekt ausgefüllten Request wird auch der Security Token zurück gesendet. Dieser kann dann auf Korrektheit geprüft werden

## Vorteil gegenüber session-basierter Authentication

Skalierbar über load-balanced Server ohne Session Sharing

### Nutzung der CSRF Middleware zur Verhinderung CSRF mit Express

```
//Enable Express csrf protection
app.use(express.csrf());

//Middleware places csrfToken in a place
//accessible by the View Rendering Engine
app.use(function(req, res, next) {
  res.locals.csrfToken = req.csrfToken();
  next();
});

//route to the secure Form calling the renderer
app.use('/securedForm.html', function(req, res, next) {
  res.render('securedForm.hbr');
}

//handlebars can access res.locals.csrfToken while rendering html
<input type="hidden" name="_csrf" value="{{ csrfToken }}>
```

Bes.  
Nutz.  
jsonwi

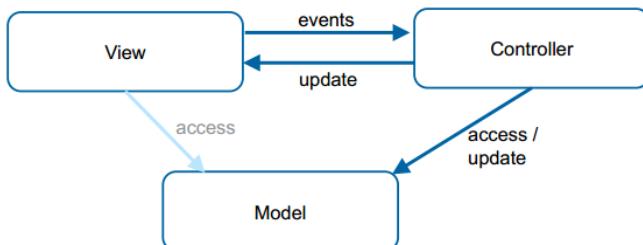
Source: <http://nodegoat.herokuapp.com/tutorial/a8>

Top Overlooked Security Threats To Node\_js Web Applications Presentation

⇒ **Separates Dokument auf den Skripteserver** (Top Overlooked Security Threats To Node\_js Web Applications Presentation 1-EXTRACT.pdf)

## Layering im Browser

Das Wichtigste Konzept im Bereich von User Interfaces ist das MVC Konzept (Model-View-Controller). Die Abbildung stellt ein mögliches Zusammenspiel von Model-View-Controller dar. Zentrale Grundregel dabei:  
Das Model hat keine statische Referenz auf den View.



### MVC Beispiel eines Counters

```

// model
const counter = {
  count: 0,
  countUp: function () {this.count++},
  countDown: function () {this.count--}
};

// view-refs
const countDisplay =
  document.getElementById("countDisplay");
const btnUp = document.getElementById("btnUp");
const btnDown = document.getElementById("btnDown");

// viewUpdate
function updateView () {
  countDisplay.innerHTML = counter.count;
}

// controller / EventListener
btnUp.addEventListener("click", function () {
  counter.countUp(); updateView());
btnDown.addEventListener("click", function () {
  counter.countDown(); updateView()));

// init
updateView();
  
```

### MVC Beispiel eines Multi-Counters (dynamic view & model creation)

```

// model
counters = [];
JS Objekt
Konstruktur

const Counter =
  function (initDelta, initCount){
    let delta = initDelta || 1;
    this.count = initCount || 0;
    this.countUp = function (){
      this.count=this.count+delta
    };
    this.countDown = function () {
      this.count=this.count-delta
    };
  };

function addCounterRow () {
  // dynamic model creation
  const currentRowNr = counters.length+1;
  const counter = new Counter(currentRowNr, 0);
  counters.push(counter);
  JS Objekt
  Instanzierung

  // dynamic view creation
  const counterRows = document.getElementById("counterRows");
  //TODO: create elements with document.createElement
  counterRows.insertAdjacentHTML('beforeend',
    '<p>Count: <span id="disp'+currentRowNr+'"></span></p>';
    '<button id="btnUp'+currentRowNr+'">Count Up</button>';
    '<button id="btnDown'+currentRowNr+'">Count Down</button>');
  counterRows.insertAdjacentHTML('beforeend',
    '<button id="btnUp'+currentRowNr+'">Count Up</button>');
  counterRows.insertAdjacentHTML('beforeend',
    '<button id="btnDown'+currentRowNr+'">Count Down</button>');

  // dynamic view reference creation
  counter.display = document.getElementById('disp'+currentRowNr);
  const uBtn = document.getElementById('btnUp'+currentRowNr);
  const dBtn = document.getElementById('btnDown'+currentRowNr);

  // dynamically attach controllers / event listeners
  uBtn.onclick = function () {counter.countUp(); updateView()};
    // could also directly update display. This would eliminate need for
    dBtn.onclick = function () {counter.countDown(); updateView()};

  updateView();
}
  
```

# OO Programmierung mit Javascript

«this» und den Konstruktoren auf den Grund gehen.

Warum?

**Objektorientierung erlaubt erhöhte Kohäsion.** Die zusammengehörigen Werte werden als Eigenschaften (Properties) im gleichen Objekt gespeichert. Funktionen die auf die Eigenschaften des Objektes arbeiten werden zusammen mit den Properties im gleichen Objekt als Methoden gespeichert. Properties und Methoden können als privat definiert werden. Sie sind dann nicht öffentlich verfügbar.

Die Objekte erweitern die Standard-Typen der Programmiersprache. Alle Instanzen einer Klasse von Objekten implementiert das gleiche API (gleiche Properties, gleiche Methoden). Unterklasse müssen das API der Oberklasse implementieren. Sie können dazu die Implementation der Oberklasse nutzen, diese aber auch überschreiben. Unterklassen dürfen das API erweitern. Methoden mit gleichem Namen, aber unterschiedlicher Signatur können unabhängig voneinander implementiert werden (Polymorphismus). Dies wird in JS nicht unterstützt.

Properties, Methoden und «This»

JS Objekte bündeln Daten als Properties.

## Bekannt

- Bekannt: Objekte als JSON definiert

```
const counter = {
  count: 0,
};

const deltaCounter = {
  count: 0,
  delta: 1
};

■ "Bekannt": Factory-Funktionen
function newDeltaCounter (cInit, dInit) {
  return {
    count: cInit || 0,
    delta: dInit || 1
  };
}

const deltaCounter2 = newDeltaCounter (0, 1);
```

- Bekannt: Zugriff auf Daten

```
console.log(deltaCounter2.count)
deltaCounter2.count = 7;
deltaCounter2['delta'] = 5;
```

- "Bekannt": Factory-Funktionen (Alternative)

```
function newCounter (cInit) {
  const o = {};
  o.count = cInit;
  return o;
}
const counter2 = newCounter (2);
counter2.count += 3;
```

## Neu

- "Möglich": Funktionen auf Objekten (extern)

```
function incCounter (cntr, delta) {
  cntr.count += delta;
}
incCounter(counter2, 22);
show("counter2.count");
incCounter(deltaCounter2, 22);
```

- Besser: Objekt mit Properties und Methoden (generiert mit Factory-Funktion) "this" notwendig

```
function newCounterWithInc (cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (delta) {
    this.count += delta
  };
  return o;
}
const counter3 = newCounterWithInc (7);
counter3.inc(5);
```

- Alternative: keine Properties, nur Methoden die auf Closure Variablen arbeiten  
-> Private Eigenschaften & Methoden

```
function
newCounterWithPrivateCountAndClosureInc
(cInit) {
  const o = { };
  let count = cInit;
  o.inc = function (delta) {
    count += delta;
  };
  o.getCount = function () {
    return count;
  };
  return o;
}
```

- Wichtig: Verständnis von

- Lexical Scope (kein dynamic Scope)
- Closure

```
function newCounterWithInc (cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (delta) {
    this.count += delta
  };
  return o;
}
const counter3 = newCounterWithInc (7);
counter3.inc(5);
```

«this» ist immer das Objekt, welches vor dem Punkt beim Aufruf der Methode steht.

«this» verhält sich wie ein impliziter Parameter der Methode. Wie alle anderen Parameter ist er lexically scoped.

Alle Aufrufe von anderen Methoden innerhalb einer Mehtode müssen mit this.methodName(..) geschehen.

Globale Funktionen können mit call (oder apply) temporär wie Methoden aufgerufen werden (this ist dann korrekt gesetzt).

```
function globalIncCountOfThisFn (delta) {
  this.count += delta;
}
function newCounterWithDelegatedInc (cInit) {
  const o = {};
  o.count = cInit;
  o.logTheCount = function () {
    console.log("log", this.count);
  };
  o.inc = function (delta) {
    globalIncCountOfThisFn.call(this, delta);
    this.logTheCount();
  };
  return o;
}
const counter7 = newCounterWithDelegatedInc(7);
counter7.inc(7); //increments count currently
show("counter7.count"); //14 -> correct
```

Achtung beim "Abtrennen" von Methoden, welche «this» nutzen

```
function newCounterWithInc (cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (delta) {
    this.count += delta;
  };
  return o;
}
const counter8 = newCounterWithInc (8);
const incFunction = counter8.inc;
global.count = 0;
incFunction(8); // increments the global var count
show("counter8.count"); //8 -> not incremented
show("global.count"); //8 -> incremented

const counter9 = newCounterWithInc (9);
const incFunction2 = counter9.inc.bind(counter9);
global.count = 0;
incFunction2(9); // increments the global var count
show("counter9.count"); //18 -> incremented
show("global.count"); //0 -> not incremented
```

Wird eine Methode vom Objekt abgetrennt aufgerufen, dann ist «this» das globale Objekt (global in Node, window im Browser).

Mit bind() kann eine ständige Bindung von Methode und Objekt erreicht werden.

Achtung in Methoden mit anonymen Funktionen

Wird in einer Methode eine anonyme Funktion aufgerufen (z.B. in Array.forEach), dann ist this z.B. der Array, und nicht das Objekt.

```

function newCounterWithArrayInc3 (cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (deltaArray) {
    deltaArray.forEach(function (arrayDelta) {
      // here 'this' is the array
      // (nothing happens)
      this.count += arrayDelta;
    });
  };
  return o;
}
const counter10 = newCounterWithArrayInc3 (10);
counter10.inc([5, 3, 2]);
show("counter10.count"); //10 -> not incremented

```

Mit bind(this) oder durch Nutzung des zweiten Parameters in der array.ForEach Funktion kann «this» korrekt gesetzt werden. Die 3. Alternative ist die Ausnutzung des Lexical Scopes und Einführung/Nutzung einer Variable that = this.

```

function newCounterWithArrayInc4 (cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (deltaArray) {
    deltaArray.forEach(function (arrayDelta) {
      this.count += arrayDelta;
    }, this);
  };
  return o;
}
const counter10a = newCounterWithArrayInc4 (10);
counter10a.inc([5, 3, 2]);
show("counter10a.count"); //20 -> incremented

```

## Vererbung & Typenerweiterung

### Primitive Types

<b>Boolean</b>	True oder false
<b>Number</b>	0 oder 1,2,3, etc.
<b>String</b>	«string»
<b>Symbol (ECMA6)</b>	
<b>Undefined</b>	Undefined, can be overwritten
<b>(null)</b>	

## Allgemeines Verhalten

Immutable, Copy by Value, Call by value und Compared by Value.

### Reference Types

■ [Object]	<b>new Object() or {}</b>	
■ Boolean	<b>new Boolean()</b>	
■ Number	<b>new Number()</b>	
■ String	<b>new String()</b>	
■ Date	<b>new Date()</b>	
■ Array	<b>new Array() or []</b>	
■ RegExp	<b>new RegExp() or /reg-exp/</b>	
■ Function	<b>new Function() or function() {}</b>	
<i>Remarks:</i>		
<i>Don't use primitive type constructors in your code!</i>		
■ null (as type, null-value is a primitive, similar to [0x000000] )		

## Allgemeines Verhalten

Objekte sind ähnlich wie Dictionaries. Jeder Referenztyp erbt vom Objekt (Object). Copy, Call und Compared by Reference. Auto-Unboxing mit .valueof().

## ■ Ablauf von 'new'

### Pseudo-Code:

```
function new(constFn) {
    newThis = {};
    constFn.apply(newThis, arguments);
    newThis.__proto__ =
        constFn.prototype;
    newThis.constructor = constFn;
    return newThis;
}
```

```
global = global || window;

function Counter(cInit) {
    if (this == global)
        throw new Error('Whoops Counter Constructor called directly');
    this.count = cInit;
    this.inc = function (delta) {
        this.count += delta;
    };
}

//const counter12a = Counter(12) //not working
const counter12 = new Counter(12);
counter12.inc(12);
show("counter12.count"); //--> 24 (expected)
show("typeof counter12"); //--> object
show("counter12 instanceof Counter"); //--> true
show("counter12.constructor.name"); //--> Counter
```

## Übersicht – Methoden für die Erstellung von JS Objekten

### ■ Object Literal (JSON)

```
var newObject = { color: "green", colorize: function() {} };
```

### ■ (( Direkte Nutzung des Object Konstruktors))

```
var newObject = new Object();
newObject.color = "green";
newObject.colorize: function() {};
```

### ■ (( Factory Function ))

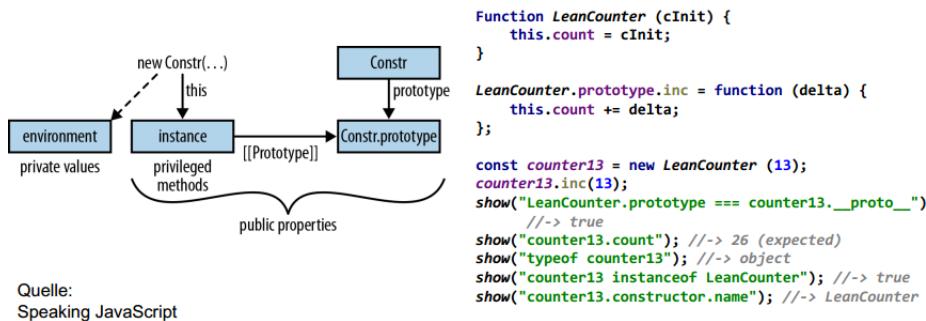
### ■ (( Für mehrstufige Vererbung: Direkte Nutzung der Object.Create Static Method ))

- Object.Create(prototype) Umgeht die Initialisierung durch den Konstruktor

## Prototypen – Objekte für die effiziente Speicherung von Methoden

Jeder Funktion (auch Nicht-Konstruktoren) wird bei der Erstellung ein leeres «Prototyp»-Objekt zugeordnet. Jedes mittels Konstruktor erstelltes Objekt hat einen Pointer auf ein «Prototyp»-Objekt. Wird eine Methode (oder Property) mittels .Operator oder [] vom Objekt angefordert, und nicht gefunden, dann wird die Suche beim «Prototyp»-Objekt weitergeführt.

Dies ermöglicht z.B. das Methoden nicht für jedes Objekt erstellt werden müssen, sondern nur einmal beim «Prototyp»-Objekt. Bei der Ausführung von beim Prototyp-gefundenen Methoden wird «this» auf das originale Objekt gesetzt.



## Private Eigenschaften

Private Eigenschaften (und Methoden) können über Closures erzielt werden. In diesem Fall müssen die entsprechenden Methoden im Konstruktor definiert werden.

Private Eigenschaften können auch über Object.defineProperty definiert werden (dies ist eine sehr mächtige Funktion, nur dieser Aspekt wird hier behandelt).

```

methoden und Konstruktor gemeinsam werden

function LeanCounterPrivate (cInit) {
  //private with closure
  let count = cInit;
  //functions functions accessing
  //count must be defined in scope
  this.getCount = function () {return count;};
  this.inc = function (delta) {count += delta;};
}

const counter13b = new LeanCounterPrivate (13);
counter13b.inc(13);
show("counter13b.getCount()");
//-> 26(expected)

function LeanCounterWithAccessor (cInit) {
  //private with closure
  let count = cInit;
  //functions functions accessing count must be
  //defined in scope
  Object.defineProperty(this, 'count', {
    // property definition with accessors
    get: function() { return count; },
  });
  this.inc = function (delta) {count += delta;};
}

const counter13c = new LeanCounterWithAccessor
(13);
counter13c.inc(13);
show("counter13c.count"); //-> 26 (expected)

```

## Beispiel

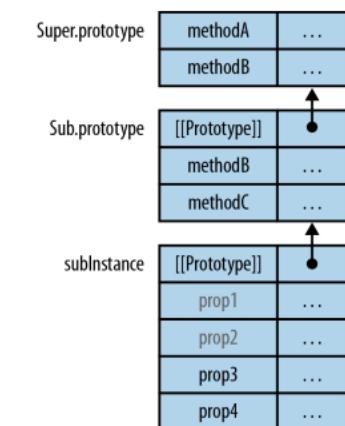
```

function House (color) {
  var self = this;
  var height = 0;
  self.facadeColor = color;
  self.paint = function(newColor) {
    repaint(newColor);
  };
  function repaint() {
    self.facadeColor = newColor;
  }
  Object.defineProperty(self, 'height', {
    get: function() { return height; },
    set: function(value) { height = Number(value); }
  });
}

```

## Mehrstufige Vererbung und die Prototype Chain

Mehrstufige Vererbung wird dadurch erreicht, dass bei erfolgloser Suche nach Methoden (und Properties) im Prototyp danach im Prototyp des Prototyps weitergesucht wird. Ist der Prototyp vom Typ des Ober-Typs (sichergestellt durch SubC.prototype = Object.create(SuperC.prototype)) Dann wird die Suche entsprechend im Prototyp des Ober-Typs weitergeführt.



Source Speaking JavaScript  
[http://speakingjs.com/es5/images/spjs\\_2107.png](http://speakingjs.com/es5/images/spjs_2107.png)

```

LeanCounter.prototype.demoInherit = function () {
  console.log("Inherited from LeanCounter");
};

function DeltaLeanCounter (cInit, dInit) {
  LeanCounter.call(this, cInit);
  this.delta = dInit;
}
DeltaLeanCounter.prototype =
  Object.create(LeanCounter.prototype);

DeltaLeanCounter.prototype.inc = function (d) {
  LeanCounter.prototype.inc.call(this, d||this.delta);
};
const counter14 = new DeltaLeanCounter (14, 14);
counter14.inc();
counter14.demoInherit(); //-> Inherited from LeanCounter
show("DeltaLeanCounter.prototype === counter14.__proto__");
show("counter14.count"); //-> 28 (expected)
show("typeof counter14"); //-> object
show("counter14 instanceof DeltaLeanCounter"); //-> true
show("counter14 instanceof LeanCounter"); //-> true
show("counter14.constructor.name"); //-> LeanCounter (known

```

## ES6 / ECMAScript 2015, Functionals JS, JS Coding Styleguides

### ECMAScript 2015 (ES6)

ES6 (ECMASCRIPT 2015) hat eine sehr gute Unterstützung durch Desktop Browser & Node. Die Unterstützung der mobilen Browser (Android) ist schlecht.

Bei der EMCAScript Version 2016+ ist eine weitgehende Unterstützung durch Desktop Browser und Node vorhanden. Die mobilen Browser sind ebenfalls schlecht unterstützt.

### Grundlagen

#### Erweiterter Syntax für Funktionsdefinitionen und Aufrufe

Funktionen können in der Parameterliste Default-Werte für Parameter definieren.

```
function paddedText(text = '', padding = '***') {
  return padding + ' ' + text + ' ' + padding;
}
```

Argumente sind keine benannten Argumente. Parameter die ausgelassen werden sollen, müssen als undefined übergeben werden damit der Default-Wert ersetzt wird. Andere falsy Werte werden nicht ersetzt.

```
log(paddedText(undefined)); // nutzt text = ''
log(paddedText(null)); // behält text = null
```

Funktionen können benannte «Rest-Parameter» definieren.

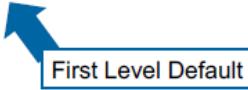
```
function sumAll(...numbers) {
  return numbers.reduce((total, next) => total + next);
}
log('sumAll(1, 2, 3)', sumAll(1, 2, 3));
```

Bei Aufrufen von Funktionen kann mit dem Spread-Operator ein Array als einzelne Argumente übergeben werden.

```
const numArray = [3, 6, 9, 12];
Log('sumAll(...numArray=[3, 6, 9, 12])', sumAll(...numArray));
```

ES6 Funktionen können Parameter für Objekt-Argumente definieren. Diese Objekt-Argumente werden zerlegt und ihre Komponenten beim Aufruf den entsprechenden Variablen zugeordnet. Default können auf der Ebene des Objekts und auf der Ebene der einzelnen Parameter definiert werden. Hierdurch lassen sich Funktionen mit «Named Arguments» definieren.

```
function carFactory({ brand = 'Volkswagen', year = 1999 } = {}) {
  return {
    brand,
    year,
  };
}
log(carFactory({ year: 2012 }));
```



Die Nutzung von var zur Deklaration von Variablen wird nicht mehr empfohlen. Variablen möglichst mit const deklarieren. Dies sind Variablen die im Folgenden nicht mehr zugewiesen werden. Eine Zuweisung...

- ... wird in WebStorm markiert und
- ... führt bei Ausführung zu einem Runtime-Type-Error
- ... von Eigenschaften von Objekten in einer const-Variable dürfen zugewiesen werden.

Variablen bei denen die Zuweisung von Werte möglich sein soll, sollten mit let deklariert werden.

#### Seitenbemerkung: Konstanten vs. Konfiguration

- Guter Code zeichnet sich durch die Einhaltung der Regel "no magic numbers" aus  
**SCHLECHT:**  
`if (width<800) { ... }`
- Magic Numbers können durch die Definition von Konstanten vermieden werden  
**BESSER**  
`const SMALL_SCREEN_WIDTH = 800;`
- Am besten ist aber die Nutzung eines Konfigurations-Objektes  
**AM BESTEN:**  
`const CONFIG = {  
 smallScreenWidth: 800, ... }`

#### Hoisting & Temporal Dead Zone (TDZ)

Die Deklaration von mit var deklarierten Variablen wird an den Anfang der umgebenden FUNKTION angehoben ("hoisted"). Die Deklaration von mit let und const deklarierten Variablen wird an den Anfang des umgebenden BLOCKS angehoben ("hoisted"). Im Gegensatz zu mit var deklarierten Variablen ist aber der Zugriff (Lesen oder Schreiben) auf diese Variablen zwischen dem Anfang des Blocks und dem Ort der Deklaration "verboten". Es resultiert ein Runtime- ReferenceError. Dieser Bereich heisst die "Temporal Dead Zone" (TDZ). WebStorm warnt wenn Variablen vor ihrer Deklaration genutzt werden.

```
function LetScope() {
  Log('letScope1 start');
  // l1 = 7; -> ref-error (TDZ)
  // Log(l1); -> ref-error (TDZ)
  let l1 = 'l1';
  Log(l1);
  Log('letScope1 end');
}
LetScope();

function LetScope2() {
  Log('letScope2 start');
  l1 = 7; //-> no ref-error
  Log(l1); //-> no ref-error
  if (true) {
    let l1 = 'l1';
    Log(l1);
  }
  Log('letScope2 end');
}
LetScope2();
```

Destrukturierung von Werten ist nicht nur bei den Funktionsparametern, sondern generell bei allen Variablendeklarationen möglich. Auch hier können Default-Werte definiert werden. Beispiel:

```
function carFactory3(optionsObj = {}) {
  const { brand = 'Volkswagen', year = 1999 } = optionsObj;
  return {
    brand,
    year,
  };
}
Log(carFactory3({ year: 2013 }));
```

#### Erweiterter Syntax für «Literale»

Literale erlauben komplexe Instanzen von Objekten und Arrays direkt zu beschreiben.

#### Bekannt

```
const a1 = [1, 'str', 3.5, { x: false
}, undefined, , true]
Log(a1); Log (a1[4], a1[4]);
Log(...a1);
```

#### Neu – Kurzschreibweise von Properties und Werten

```
const x = true;
const y = 'str2';
const z = -23.8;

console.log( { x, y, z } );
```

#### Neu – Berechnete Eigenschaftennamen in Literalen

```
function createEntryES6(name, keyValue,
  keyName = 'key') {
  return {
    name,
    [keyName]: keyValue
  };
}
const entryES6 = createEntryES6('markus', 54, 'age');
```

#### Bisher – Mehr Code nötig.

```
function createEntryES5(name, keyValue, keyName) {
  var keyN = keyName || 'key';
  var newEntry = {
    name: name
  };
  newEntry[keyN] = keyValue;
  return newEntry;
}
```

Template String starten und enden mit einem «Backtick» ` = Rückwärts geneigtes Hochkomma. Die Template Strings dürfen \${exp} enthalten. Diese werden evaluiert und deren String-Präsentation am Ort eingefügt.

```
log('simple template string',
`The result of 2+3 equals ${2 + 3}`);

function createSongsHTML(songsArray) {
  return `<ul>
${songsArray
  .map(song => `<li>${song.title} by ${song.artist}</li>`)
  .join('\n    ')
}
</ul>`;
}
```

### Einfacher Syntax für Getters & Setters Methoden in Object Literalen

ES6 erlaubt die einfache Definition von Property getters, setters und Methoden direkt in Object-Literalen (Objekt-Instanzen).

```
function eS6LeanCounterFactory(initCount) { function LeanCounterWithAccessor(cInit) {
  let myCount = initCount;
  return {
    get count() {
      return myCount;
    },
    set count(newCount) {
      myCount = newCount;
    },
    inc(delta) {
      myCount += delta;
    },
  };
}

let count = cInit;
Object.defineProperty(this, 'count', {
  get() {
    return count;
  },
  set(newCount) {
    count = newCount;
  },
});
this.inc = function (delta) {
  count += delta;
};
```

Abbildung 1: Entsprechender E5 Code

### Arrow Funktionen enthalten «this» aus dem Kontext

Mit ES6 Array Funktionen wird automatisch das «this» des Kontexts genutzt.

```
function newES6CounterWithArrayInc(cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (deltaArray) {
    deltaArray.forEach((arrayDelta) => {
      this.count += arrayDelta;
    });
  };
  return o;
}
```

#### ■ Entsprechender ES5 Code

```
function newCounterWithArrayInc4(cInit) {
  const o = {};
  o.count = cInit;
  o.inc = function (deltaArray) {
    deltaArray.forEach(function (arrayDelta) {
      this.count += arrayDelta;
    }, this); // Alternative: use fn(){}.bind(this)
  };
  return o;
}
```

Die ES6 Classes Syntax bietet eine einfach zu verstehende Schreibweise um die entsprechenden JS Konstruktoren zu definieren. Daher mit einer Class Definition wird ein entsprechender Konstruktor mit speziellen Eigenschaften angelegt. Speziell:

- Aufruf ohne new führt zu einem Type-Error
- Die Deklaration wird nicht angehoben => new kann nicht aufgerufen werden vor der Deklaration

Die Verbindungen des generierten Constructors, desses Prototyp und des Prototyps von Instanzen entspricht der bei ES5. Properties müssen "manuell" in der constructor() Methode erstellt werden. Diese sind dann enumerable direkt auf den Instanzen gespeichert. Methoden werden nicht-enumerable auf dem Prototyp-Objekt für alle Instanzen gemeinsam zugreifbar abgelegt.

Dadurch dass Methoden in einem von der constructor() Methode unabhängigen Scope definiert werden, können diese Methoden nur auf Properties des Objekts, aber nicht auf gemeinsame Variablen in einem gemeinsamen Scope (Closure) zugreifen.

```
class Counter {
  constructor({
    start: start = 0,
    step: step = 1 } = {}) {
    this._count = start;
    this._step = step;
  }
  get count() {
    return this._count;
  }
  inc(step = this._step) {
    this._count += step;
  }
  dec(step = this._step) {
    this._count -= step;
  }
}
const c2 = new Counter();
c2.inc(7);
log('c2.count', c2.count);
show('ES6 Classes behave mostly like normal JS Constructors');

show('Counter.prototype === c2.__proto__'); // -> true
show('Counter.prototype === Object.getPrototypeOf(c2)'); // true
log('ES6 classes store methods in the prototype, but non-enumerable');
show('Counter.prototype'); // -> shows no visible methods
show('Object.getOwnPropertyNames(Object.getPrototypeOf(c2))');
show('Object.getOwnPropertyDescriptor(c2, "_step")');
show('typeof c2'); // -> object
show('c2 instanceof Counter'); // -> true
show('c2.constructor.name'); // -> Counter
```

Die ES6 Class Definition Syntax vereinfacht die Definition von Unterklassen. In der Constructor Methode (wenn definiert) sollte super() aufgerufen werden. In anderen Methoden kann auch super.Methode aufgerufen werden um auf die Methoden der Oberklassen zuzugreifen.

```
class DoubleCounter extends Counter {
  constructor({ start: start = 0, step: step = 1 } = {}) {
    super({ start, step });
    this._step = this._step * 2;
  }
}

const dc = new DoubleCounter({ step: 4 });
dc.inc();
show('dc'); -> 8
```

## Funktionales Javascript

### Theorie

JS funktional zu schreiben ist ein spezieller Stil. Besonders sichtbar wird er durch das Function-chaining bei dem Zeilen mit einem Methodenaufruf nicht mit ; enden und die nachfolgende Zeile mit

## Web Engineering + Design 2

.<methodenAufruf> weiterfährt. Wir haben Beliebt ist der funktionale Stil zum Beispiel im Zusammenhang mit der Verarbeitung von Strings und Arrays. Hier eignen sich die folgenden Funktionen «Chaining»:

```
Array.from(Iterable) generiert einen Array z.B. aus einem String
Array.filter(checkIfAllowedBoolFn (element [, index]) )
Array.map(elementTransformFn (element, [index]) -> Wendet fn auf jedes Element des Arrays an
Array.join(separatorString) -> Konkatiniert alle Elemente eines Arrays in einen String
Array.reduce(combineFn(prevResult, element, currentIndex, array), initialValue)
-> Kombiniert Schritt-für-Schritt paarweise das jeweils aktuelle Element mit dem bisherigen Resultat
```

### Beispiele

#### One

```
const str1 = 'hallo hallo hallo';

const str2 = Array.from(str1)
  .filter(c => 'abcdefghijklmnopqrstuvwxyz'.indexOf(c) < 0)
  .map(c => c + c)
  .join('');

console.log(str2); // -> llllloo  llllloo  llllloo

const str3 = Array.from(str1)
  .filter(c => 'abcdefghijklmnopqrstuvwxyz'.indexOf(c) < 0)
  .map(c => c + c)
  .reduce((resString, c) => resString+c, '');

console.log(str3); // -> llllloo  llllloo  llllloo
```

#### Two

Convert a string to a new string where each character in the new string is '(' if that character appears only once in the original string, or ')' if that character appears more than once in the original string. Ignore capitalization when determining if a character is a duplicate.

Examples:

```
"din" => "((("  

"recede" => "()())"  

"Success" => ")()())"  

"(( @)" => ")()("
```

```
function duplicateEncode(word){  

  return word  

    .toLowerCase()  

    .split('')  

    .map( function (a, i, w) {  

      return w.indexOf(a) ==  

        w.lastIndexOf(a) ? '(' : ')'  

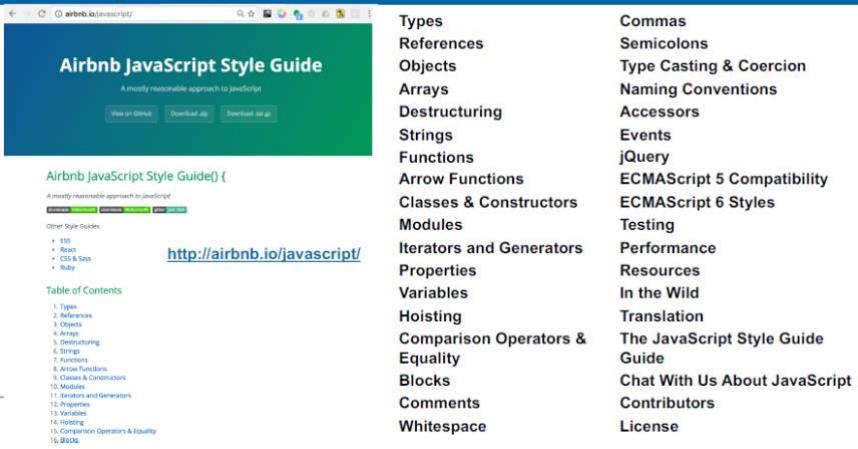
    })  

    .join('');
}
```

Source:

<https://www.codewars.com/kata/duplicate-encoder/javascript>

## JS Styleguides



The screenshot shows the Airbnb JavaScript Style Guide website. The top navigation bar includes links for "View on GitHub", "Download zip", and "Download tar.gz". The main content area is titled "Airbnb JavaScript Style Guide" and describes it as "A mostly reasonable approach to JavaScript". It features a "Table of Contents" on the left and two columns of style rules on the right. The left column includes sections like Types, References, Objects, Arrays, Destructuring, Strings, Functions, Arrow Functions, Classes & Constructors, Modules, Iterators and Generators, Properties, Variables, Hoisting, Comparison Operators & Equality, Blocks, Comments, and Whitespace. The right column includes sections like Commas, Semicolons, Type Casting & Coercion, Naming Conventions, Accessors, Events, jQuery, ECMAScript 5 Compatibility, ECMAScript 6 Styles, Testing, Performance, Resources, In the Wild, Translation, The JavaScript Style Guide Guide, Chat With Us About JavaScript, Contributors, and License.

Viele Regeln des Airbnb JS Styleguides können mit ESLint automatisch überprüft werden. Dies ist auch direkt in Webstorm möglich.

## Installation

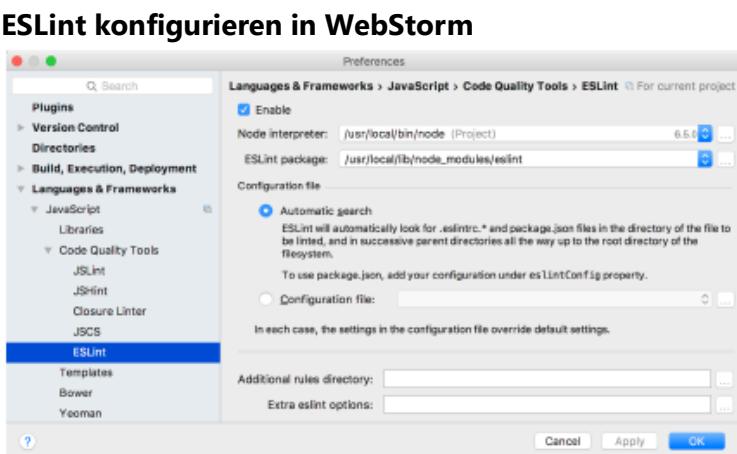
```
npm install eslint --global
npm install eslint eslint-config-airbnb --global
npm install eslint-plugin-jsx-a11y@^2.0.0 eslint-
plugin-react eslint-plugin-import babel-eslint -
global
```

File `.eslintrc` erstellen in user root oder anderer Stelle oberhalb des Projektfolders.

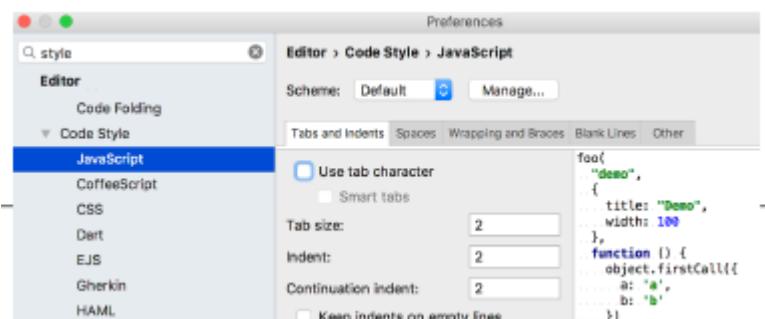
Inhalt:

```
{
  "extends": "airbnb",
  "rules": { "func-names": ["error", "never"] }
}
```

ESLint konfigurieren in WebStorm



## Coding Style adaptieren (+ reformat + autofix)



## JS Module

### Module Theorie

#### Definition

Ein (JS)-Modul ist eine Gruppierung von Programm-Code, die in einem File verwaltet wird. Die eine eigenen Scope besitzt und damit Namenskonflikte mit anderen, im Kontext parallel definierten Modulen vermeiden kann. Es stellt den Nutzern des Modules Objekte und/oder Funktionen zur Verfügung, die Sie in ihren Namensraum importieren können. Normalerweise wird es separat vom nutzenden Programm geladen.

#### Vorteile von Modulen

- Ermöglicht parallele System-(weiter)-Entwicklung durch mehrere Personen wird einfacher
- Ermöglicht Kapselung von wiederverwendbaren Funktionalität in einem File («high cohesion» innerhalb des Moduls, «low coupling» mit anderen Modulen)
- Ermöglicht «lazy loading», daher schnelle Start-Up Zeiten, da nicht der gesamte Code direkt beim Programm-Start geladen werden muss

Es gibt viele Informatik Begriffe die Ähnlichkeiten zum Begriff «Modul» aufweisen.

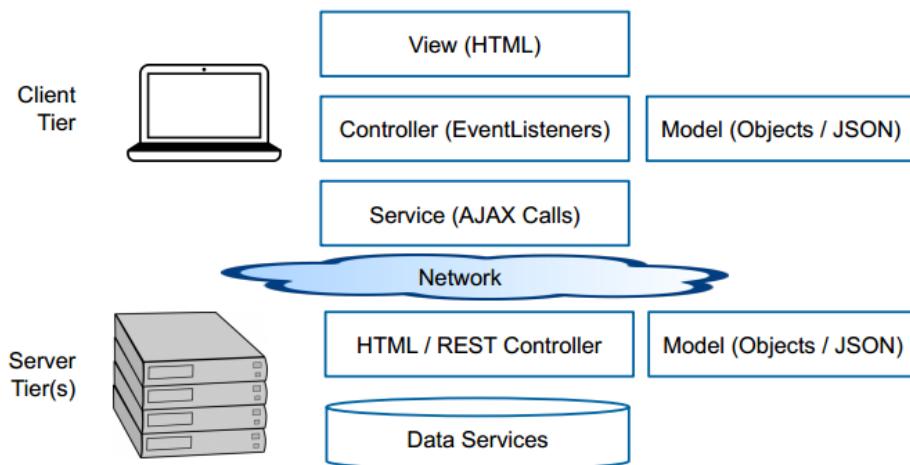
	Controller	Modul	View	Physikalische Aufteilung / Zusammenfassung			Package	Modul
File				File	Folder	Tier		
Klasse	Objekt	Tier	Package	Logische Aufteilung / Zusammenfassung (meist mit API / Information hiding)			Klasse	Objekt
Unit (Unit Test)		Layer		Folder	Unit (Unit Test)	(Web)-Componente	Layer	
Model	(Web)-Componente		Service	Beschreibung der Funktion einer Gruppe von Code			Model	View
				Controller			Controller	Service

Teil der JS Sprachdefinition  
zur Code-Strukturierung

### Typische Konventionen bei JS-Modulen

- 1 File = 1 Modul = 1 Unit (mit assoziierten Unit-Tests)
- 1 Folder = 1 Package = Sammlung von «zusammengehörenden» Modulen und Klassen

### Tiers vs. Layers vs. Classes vs. Services vs Modules / Packages



### JS-Module: Nutzung auf dem Node-Server (CommonJS)

Module entsprechen einem JS-File, welches auf dem Server zugreifbar ist.

#### Einbinden von Server-Modulen

```
<newVariableForModuleExportObject> = require(Modulename)
Z.B.
CounterJsonAPI = require('./CounterJsonAPI')
>
{ newCounterJSON: [Function: newCounterJSON], countUp: [Function: countUp], countDown: [Function: countDown] }
... ...
```

#### Definition von Server-Modulen

Node-Files/Node Module haben per Definition ihren eigenen Scope, daher ist «global namespace pollution» kein Problem. Das zu exportierende Objekt wird der speziellen globalen Variable `module.exports` zugewiesen.

```
module.exports = { newCounterJSON, countUp, countDown };
```

### JS-Module: Ansätze für Module im Browser (CommonJS)

#### Revealing Module Pattern

Wird das «Revealing Module Pattern» eingesetzt, können im Browser entsprechend vorbereitete Files ohne «global namespace pollution» geladen werden.

## Einbinden von Modulen entsprechend dem «Revealing Module Pattern»

Das entsprechende File wird regulär über ein script-Element im HTML geladen.

```
<script src="scripts/Counter.js"></script>
```

Nach Konvention wird im Modul ein globales Objekt mit Namen entsprechend dem Modulnamen als Export-Objekt angelegt. Im Code der das neue Modul nutzt wird dieses Objekt entsprechend genutzt. z.B im main.js (welches in einem Script-Tag nach Counter.js geladen wird)

```
const counter = new window.Counter();
```

## Definition von Modulen entsprechend dem «Revealing Module Pattern»

Im File wird ein local-scope mittels einer iife generiert. Das Export-Objekt wird einem der globalen Variablen window zugewiesen.

```
(function () {
  class Counter { ... }
  window.Counter = Counter;
})();
```

Beim Einsatz des «Revealing Module Pattern» kann es Konflikte im Global-Namespace geben wenn zwei Module den gleichen Namen haben. Diese Konflikte können durch die Nutzung von Package-Objekten umgangen werden.

## Einbinden von Modulen mit Package-Objekten

Das entsprechende File wird regulär über ein Script-Element im HTML geladen.

```
<script src="scripts/game/Counter.js"></script>
```

Nach Konvention wird im Modul im globalen Objekt mit dem Namen des Packages ein Objekt mit Namen entsprechend dem Modulnamen als Export-Objekt angelegt. Im Code der das neue Modul nutzt wird dieses Objekt entsprechend referenziert.

```
const counter = new window.game.Counter();
```

## Definition von Modulen mit Package-Objekten

Im File wird ein local-scope mittels einer iife generiert. Das export-Objekt wird dem globalen Package-Objekt zugewiesen. Wenn nicht vorhanden, muss das Package-Objekt erst erstellt werden.

```
(function () {
  class Counter { ... }
  window.game = window.game || {};
  window.game.Counter = Counter;
})();
```

## Nutzung von RequireJS

RequireJS ist eine Library die Code im Browser asynchron lädt. RequireJS implementiert die AMD (Asynchronous Module Definition) Spezifikation.

## Einbinden von Modulen mit RequireJS

Wenn RequireJS zum Laden von Modulen genutzt wird darf nur ein einziges Script-Tag genutzt werden um alle Module zu laden. Dieses Script-Tag lädt die require.js Library und hat einen Eintrag auf das main file (Name frei wählbar). Diese main-File ist der Ausgangspunkt für alle weiteren Module-Ladevorgänge.

---

Z.B. `<script data-main="scripts/mainCombo" src="scripts/require.js"></script>`

Beispiel:

```
require(['./game/main'], function () {}); require(['./shop/main'], function () {});
```

Eine bestimmte Reihenfolge des Modulladens lässt sich durch Schachtelung der Require-Aufrufe erreichen. Es ist auch möglich Module erst später, 'on-demand' zu laden. Auch bei RequireJS wird mit Export-Objekten gearbeitet. Das Expert-Objekt eines mit require angeforderten Moduls wird als Argument der Callback Funktion im Require-Aufruf mitgegeben.

```
require ['./game/Counter', function (Counter) {
    const counter = new Counter();
    ...
}];
```

### Definition von Modulen mit RequireJS

Von RequireJS gelandene Files sollten generell (wie beim "revealing modul pattern") nicht im global-Namespace agieren, sondern eine iife öffnen um global-namespace-pollution zu verhindern. Bei Modulen die ein Export-Objekt zur Verfügung stellen, sollte am Start des Files die durch requireJS definierte Funktion define aufgerufen werden.

**Syntax: `define([ <requiredModuleIDString>,* ], <functionReturningExportObject>)`**

Define hat 2 Argumente. Das erste gibt einen Array von Modul-Namen an die geladen sein müssen bevor dieses Modul geladen werden kann. Ist der Array leer, kann dieser Parameter auch weggelassen werden. Der 2. Parameter ist die Funktion die den Code dieses Moduls enhält. Der Return-Wert dieser Funktion ist das Export-Objekt welches in die Callback-Funktion von require eingespeist wird. Diese Objekt wird von requireJS gespeichert. D.h. ein zweiter Aufruf von require[moduleName] wird ohne nochmaliges Laden beantwortet.

### Beispiel

```
game/Counter.js exportiert die Counter Klasse
define([], function () {
    class Counter { ... }
    return Counter;
});
```

### Laden von Files die Aktionen bei Window.onload registrieren

Zum Teil werden mit RequireJS Files geladen in denen JS-Code erst dann ausgeführt werden darf wenn der DOM-Tree fertig aufgebaut ist. Dies ist typischerweise der Fall wenn DOM Manipulationen nötig sind, oder Event-Handler an Elemente angehängt werden sollten. Ohne requireJS würde eine Funktion mit den Aktionen als Window.onload Event-handler registriert werden.

Es kann aber sein, dass zum Zeitpunkt des Laden eines Files mit requireJS der DOM Tree schon fertig aufgebaut ist, und somit der Window.onload Event nicht mehr feuert nachdem das File fertig geladen wurde.

Hier ist es nötig die Aktionen in eine Funktion initPage auszulagern, und diese bei komplett geladenem DOM sofort auszuführen, oder aber als onload-Event-Handler zu registrieren:

```
function initPage () { ... };
if (document.readyState === 'complete') {
  initPage();
} else{
  window.addEventListener('load', initPage);
}
```

### Nutzung ES6 Import/Export

ES6 definiert eine Syntax für die Definition von Modul-Exports und das Einbinden von Modulen.

#### Einbinden von Modulen

Mit dem Import Statement kann definiert werden, dass Module eingebunden werden sollen und in welcher lokalen (const) Variable des Export-Objekt abgelegt werden soll.

Syntax: `import <moduleExportLocalName> from <ModuleName>`

Beispiel: `import Counter from './Counter';`

#### Definition von Modulen / Definition von Modul-Exports

Mit dem Export-Statement kann definiert werden, dass ein File als Modul geladen werden kann und dass ein Export-Objekt geliefert wird.

Syntax: `export default Counter;`

Beispiel: `export default Counter;`

Der in ES6 definierte Syntax für Module wird noch von keinem Browser direkt unterstützt. Mit WebPack (einem Build-Tool) ist es möglich mittels eines Pre-Compile Schritts im Browser ladbares Javascript aus Files zu erstellen. WebPack nutzt Babel (ein ES6 zu ES5 Processor) für die Eliminierung der Import/Export Statements. WebPack kann so konfiguriert werden, dass aller Code in einem Files zusammengefasst wird. Dies ist- gegeben den aktuellen Stand der http2 Implementation sinnvoll.

## TypeScript

### Motivation und Grundlagen

Fehler passieren schnell mit Javascript, wie untenstehendes Beispiel zeigt.

#### JS Calc Demo

```
window.addEventListener('load', function () {
    // viewRefs
    var calcBtn = document.getElementById('calcBtn');
    var s1Input = document.getElementById('s1');
    var s2Input = document.getElementById('s2');
    var sumDiv = document.getElementById('sumdisplay');

    Summand 1: _____
    Summand 2: _____
    Summer berechnen
    Summe: _____
    12

    function sum (s1, s2) {
        return s1 + s2;
    }

    calcBtn.addEventListener('click', function () {
        sumDiv.innerHTML = parseInt(s1Input.value, s2Input.value));
    });
});
```

Die Fehler lassen sich einfach vermeiden mit TypeScript Static Type Checking.

```
window.addEventListener('load', function () {
    // using global ID Elements for viewRefs

    let calcBtn = document.getElementById('calcBtn');
    let s1Input = <HTMLInputElement>document.getElementById('s1');
    let s2Input = <HTMLInputElement>document.getElementById('s2');
    let sumDiv = <HTMLInputElement>document.getElementById('sumDisplay');

    function strictSum(s1: number, s2: number) : number {
        return s1+s2;
    }

    calcBtn.addEventListener('click', function () {
        sumDiv.innerHTML = strictSum(s1Input.value, s2Input.value); // error
        // sumDiv.innerHTML = strictSum(s1Input.value, s2Input.value) // a correct line
    });
});
```

TS2345:Argument of type 'string' is not assignable to parameter of type 'number'.

TypeScript ist die «Rettung» für alle denen das «Dynamic Typing» von Variablen in JS zu «dynamisch» ist. Es erlaubt die Spezifikation von Typen für Variablen, Funktionsparameter, Funktionswerte und mehr.

TypeScript erweitert den JS (ES5, 6, 7) Syntax. Weiter ist es abwärtskompatibel» zu JS: (fast) jedes korrekte JS ist auch korrektes TS. Es bietet «Intellisense» (static type checking) in unterschiedlichen Editoren.

Grundsätzlich ist er ein Pre-Compiler der JavaScript generiert (ohne Runtime-Checking).

#### Details – Typen und Syntax-Module

##### Variablen Deklarationen mit Basistypen

Variablen in Typescript-Code benötigen keine Typendeklaration. In diesem Fall nutzt Typescript "Type-Inferenz" um den Typ der Variable zu bestimmen. Variablen können explizit als any deklariert werden, dann können ihnen beliebige Werte zugewiesen werden. Variablen mit Typ any dürfen auch beliebigen anderen Variablen zugewiesen werden. Mit declare let ... kann dem TypeScript Complier mitgeteilt werden das es weitere Variable gibt. Aber es wird kein wentsprechender JS Code generiert. Basis-Typen: boolean, number, string (sowie null und undefined).

#### Beispiel

```
Beispiele
let myAnyVar1: any = 1;
                // allowed
                myAnyVar1 = 'hi';
                myNumberVar = myAnyVar1; // might come as surprise

let myInferredNumVar1 = 1;
                // not allowed
                myInferredNumVar1 = 'hi';
                myStringVar = myInferredNumVar1;
                myStringVar = 1;
                myNumberVar = 'hi';
```

TypeScript erlaubt die Deklaration von Arrays und Tupels. Bei Tupeln wird keine Type-Inferenz angewendet. Enums können wie Basistypen genutzt werden.

## Beispiel

```

Beispiel
-----  


```

let myInferredNumArray = [1, 2, 3];
let myNotInferredTupel = [1, 'abcd'];
let myNumArray:number[] = [1, 2, 3];
let myTupel: [number, string] = [1, 'abcd']
// allowed
myNotInferredTupel[0] = 2;
myNotInferredTupel[1] = 2; // no inferred
myTupel[1]='hi';

//not allowed
myInferredNumArray[2] = 'hi';
myInferredNumArray[4] = 'hi';
myTupel[0]= 'hi';
myTupel[1]= 2;

enum Color {Red, Green, Blue};
let c: Color = Color.Green;
let myTupel2: [Color, number] =
    [Color.Green, 1];

```


```

## Funktionsdeklarationen

TypeScript erlaubt die Deklaration von Funktionen. Hierbei können sowohl die Typen von Parametern als auch der Typ des Rückgabewertes spezifiziert werden. Funktionsparameter können auch optional sein, und es können für eine Funktion mehr als eine erlaubte Signatur definiert werden.

## Beispiel

```

function sumFunction(n1: number, n2: number): number {
    return n1+n2;
}

function combineFunction2(sn: number|string='', ns?: number|string): string {
    return sn.toString()+(ns || '').toString();
}
// allowed
let myNum: number = sumFunction(1, 2);

// not allowed
let myStr: string = sumFunction(1, 2);
sumFunction(1, 'hi');

```

## Funktionen als Parameter

Funktionsparameter können auch Funktionen sein. Die Signatur dieser Parameter kann auch deklariert werden.

## Beispiel

```

function numberApplicator (numArray: number[],
    numFun: (prevRes: number, current: number) => number) {
    return numArray.reduce(numFun);
}

let myNum2: number = numberApplicator(
    [1, 2, 3, 4], sumFunction);

// not allowed
let myStr: string = sumFunction(1, 2);
sumFunction(1, 'hi');
function concatFunction(s1: string, s2: string): string {
    return s1+s2;
}
numberApplicator([1, 2, 3, 4], concatFunction);

```

## Komplexe Typen – Klassen

TypeScript stellt eine ES6-erweiternde Syntax zur Definition von Klassen zur Verfügung. Properties der Instanzen und der Klasse (Static) werden im Kontext der Klasse definiert. Methoden und Properties können mit den Zusätzen "private" und readonly versehen 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 (newDoorCount >= Counter.MIN_DOOR_COUNT && newDoorCount <= Counter.MAX_DOOR_COUNT) {
            this._doors = newDoorCount;
        } else {
            throw `Counter can only have ...`;
        }
    }
    get doors() {
        return this._doors;
    }
}

```

12

## Klassen und Interfaces

Klassen können in der Konstruktor-Argumentenliste angeben ob ein Parameter zu einem public oder private Property gemacht werden soll. In diesem Fall kann sich der Code zur Initialisierung gespart werden.

TypeScript erlaubt die Definition von Interfaces. In der Deklaration von Klassen kann angegeben werden, dass diese Klasse eine Super-Klasse "extended" und ein oder mehr Interfaces 'implementiert'. Auch bei der Deklaration von Variablen und Funktionsparametern dürfen Interfaces genutzt werden.

Casting ist möglich und Structural-Typing ("Duck-Typing") wird von TypeScript unterstützt.

```

interface Point {
    readonly x: number;
    readonly y: number;
}
interface LikableItem {
    likes?: number;
}
class DescribableItem {
    constructor(public description: string){
    }
}
class PointOfInterest extends DescribableItem implements Point, LikableItem {
    constructor(public x: number, public y: number, description: string, public likes?:number) {
        super(description);
    }
}

//allowed
let p:Point = new PointOfInterest(1, 2, 'home');
let p2:PointOfInterest = <PointOfInterest>p;
p2.description
let p3:Point = { x: 3, y: 4}; // structural-typing
let p4: any = p3;
p4.description = 'hi'; // any can set anything

// not allowed
p.description

```

13

Typescript orientiert sich am ES6 Modul-Syntax. Das heisst mit Import und Export.

```
File Counter.ts
export default class Counter //export aus einem Modul
File counterDetails.ts
import Counter from './Counter';
```

Im config.ts File kann angegeben werden ob das Ziel-System der Browser und damit AMD (RequireJS) oder der Server und damit CommonJS (Node) sein soll. Zudem kann der gesamt Code auch direkt von Typescript in ein einzelnes File zusammengefügt werden. Zusammenführung von Modulen kann aber auch einem separaten Tool in der Build-Pipeline überlassen werden (z.B. WebPack).

Für die Nutzung von Modulen die nicht mit TypeScript entwickelt wurden, ist es im möglich die von diesen Modulen exportierten APIs mit "any" zu deklarieren. Damit hat man bei der Nutzung dieser APIs die vollständige Freiheit. Der Nachteil ist aber, dass die Typensicherheit verloren geht. Deswegen lohnt es es sich bei der Nutzung von Standard-Modulen wie z.B. JQuery oder Lowdash entsprechende Type-Definition Files (Endung d.ts) von DefinitelyTyped zu laden (mehr dazu hier: <http://definitelytyped.org/>)

## Abschluss

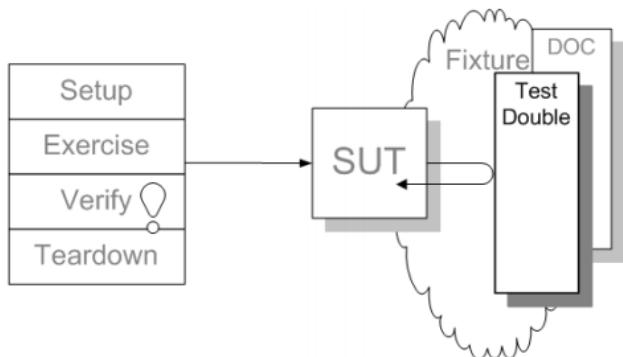
Man soll Typescript nutzen. Mit Angular2 ist dies fast unumgänglich. Als Alternative würde sich das Tool Flow von Facebook nutzen lassen (Tool zur statischen Codeanalyse).

# Testing

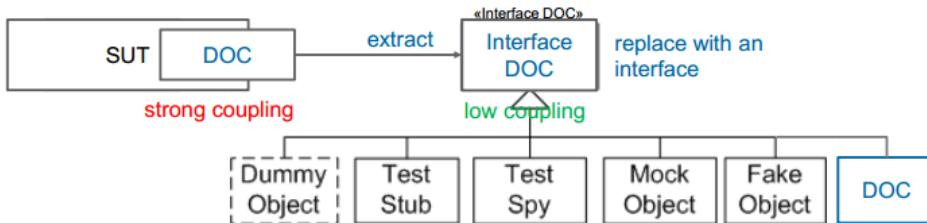
## Unit Test Patterns

### Test Double Pattern

Wie können wir die Logik isoliert testen, wenn der Code hängt davon ab, ist unbrauchbar? Wie können wir langsame Tests vermeiden?

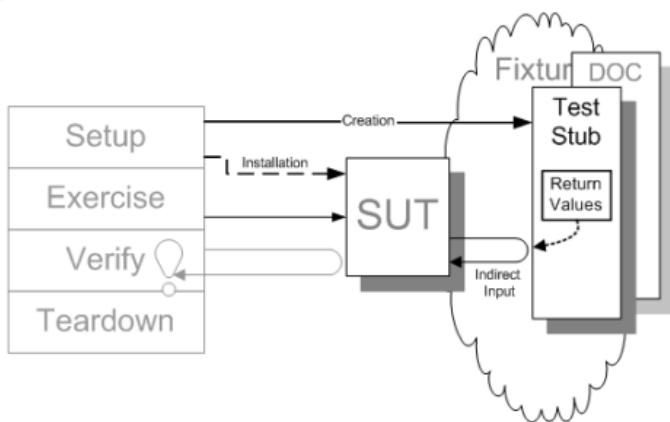


Wie können wir DOC durch ein Test Double ersetzen?

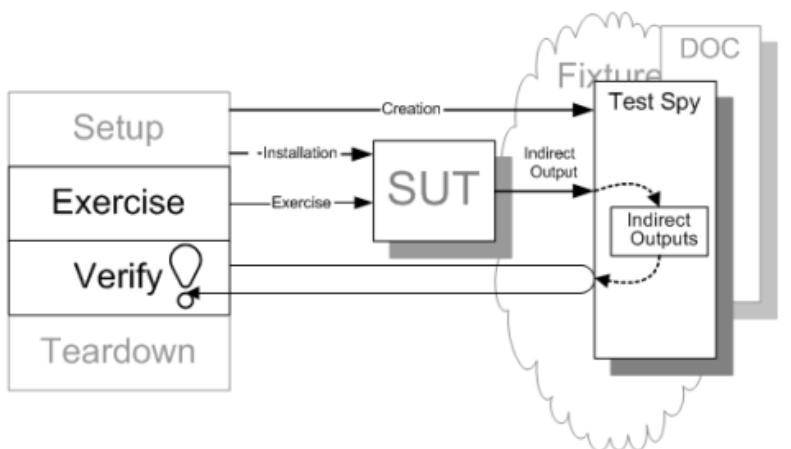


### Test Stub Pattern

Wie können wir die Logik unabhängig überprüfen, wenn sie von indirekten Eingaben von anderen Softwarekomponenten abhängig ist?

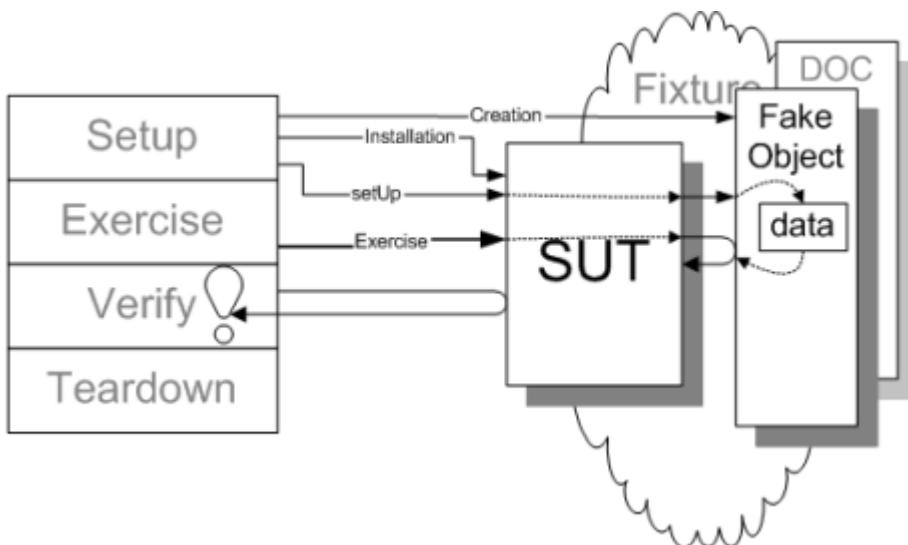


Wie können wir die Logik unabhängig überprüfen, wenn sie von indirekten Eingaben von anderen Softwarekomponenten abhängig ist?



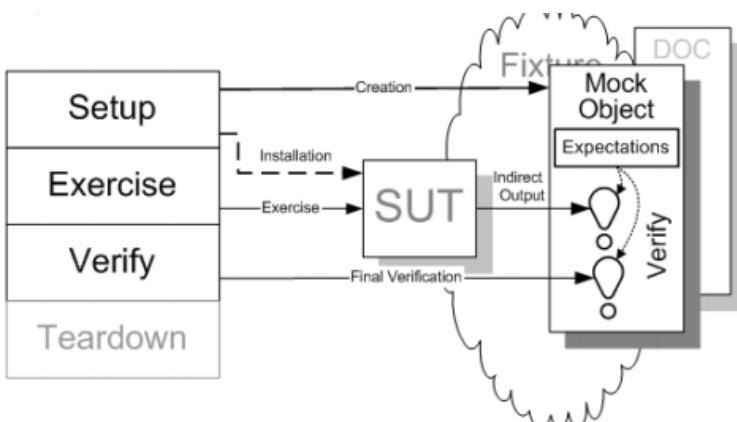
### Fake Object Pattern

Wie können wir die Logik unabhängig verifizieren, wenn abhängige Objekte nicht verwendet werden können? Wie können wir langsame Tests vermeiden?



### Mock Object Pattern

Wie implementieren wir Verhaltensüberprüfung für indirekte Ausgänge des SUT? Wie können wir die Logik unabhängig überprüfen, wenn sie von indirekten Eingaben von anderen Softwarekomponenten abhängig ist?



Basierend auf der ubiquitären Sprache der Behavior-Driven Development (BDD). BDD ist eine Erweiterung vom Test-Driven Development.

- ...Wie das gewünschte Verhalten festgelegt werden soll.
- ...Dass Business Analysts und Entwickler in OOA / OOD zusammenarbeiten sollten, um Verhalten in Bezug auf User Stories festzulegen.

### BDD User Stories

**Struktur** Erzählung sowie Akzeptanzkriterium oder User Stories

### Beispiel

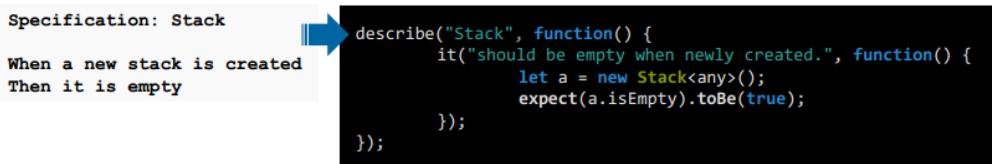
```
Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned.

Scenario 1: Refunded items should be returned to stock
Given that a customer previously bought a black sweater from me
And I have three black sweaters in stock.
When he returns the black sweater for a refund
Then I should have four black sweaters in stock.
```

### BDD Story versus Specification

Die Unterkategorie der BDD-Werkzeuge verwendet Spezifikationen als Eingabesprache und nicht als Benutzergeschichten. Jasmine stellt ein Spezifikationswerkzeug dar. Verwendet keine User Stories als Eingabeformat. Verwendet eher funktionale Spezifikationen für Einheiten, die getestet werden. Spezifikationen haben oft mehr technischen Charakter.

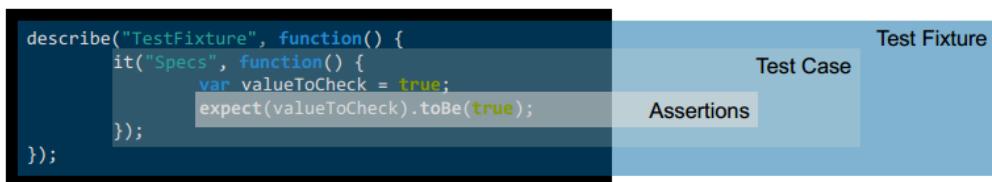


```
Specification: Stack
When a new stack is created
Then it is empty
```

```
describe("Stack", function() {
    it("should be empty when newly created.", function() {
        let a = new Stack<any>();
        expect(a.isEmpty).toBe(true);
    });
});
```

### Jasmine API

Bietet grundsätzliche Möglichkeiten, BDD-Tests zu schreiben. Die Testsuite (Testgerät) beginnt mit einem Aufruf der globalen Jasmine-Funktion describe(). Specs (Testfälle) werden durch Aufruf der globalen Jasmine-Funktion it () definiert. Erwartungen (Assertionen) werden mit der Funktion expect () erstellt, die einen Wert annimmt.



Bietet umfangreichen Satz von Matcher. Matcher bieten einen booleschen Vergleich zwischen dem Istwert und dem Erwartungswert.

```
describe("TestFixture", function() {
  it("Specs", function() {
    var valueToCheck = true;
    expect(valueToCheck).toBeGoofy();
  });
});
```

## Test-Driven Development

### Einführung

Es braucht enorme Übung um die Test zu schreiben, bevor der Implementierungscode geschrieben wird.

### Test-Driven Development ist...

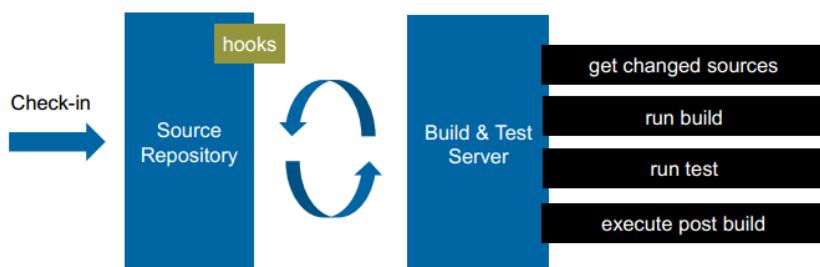
- Ein Weg um Unit Tests zu schreiben
- Ein Feedback um den Code zu validieren
- Ein umfangreicher Weg um Defekte in der Software zu vermeiden.
- Ein Design Prinzip.

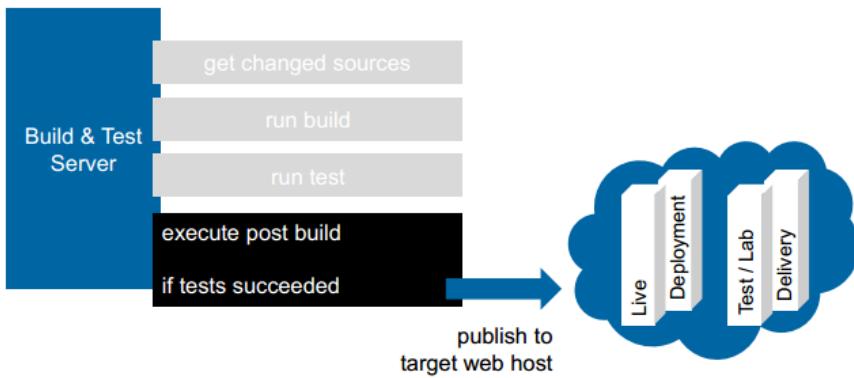
### Der Zyklus



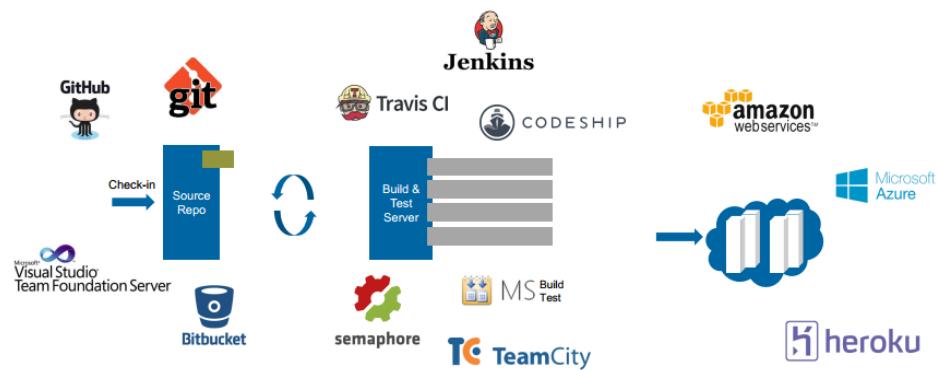
## Test-Automation

### Continuous Integration





### Mögliche eingesetzte Tools



### Tool Chain für Angular 2

