WED2 Cheatsheet 2016

Node.js

Module

```
var counter = 0; // myCounter.js
function add() {return ++counter;}
function get() {return counter;}
module.exports = {count: add, get: get };
var myCounter = require('.\myCounter.js');
```

Resolve: Core Module > File / Directory > node_modules
Zyklen durch require() möglich -> Vermeiden

Default API

- **Callback** ist immer das letzte Argument
  - Wirft keine Exceptionl
  - Erstes Argument ist der Error
- **synchrone** Methoden werfen Exceptions

Events

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('event', (a, b) => {
  console.log(a, b, this); // no this
});
myEmitter.emit('event', 'a', 'b');
```

Express

```
var http = require('http');
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var router = express.Router();
app.use(express.static(__dirname+ '/public'));
app.use(bodyParser.urlencoded({ extended: false}));
app.use(router);
http.createServer(app).listen(3000);
// alt.: function (request, response) {…}
```

Middleware

Reihenfolge der Registrierung bestimmt die Ausführreihenfolge.

```
app.use([path,] callback [, callback...])
```

Middleware mit vier Parameter gelten als Error-Handler.

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

Custom Middleware besitzt drei Parameter (request, response, next). Mit next() wird die nächste Middleware aufgerufen. Anfrage kann vorher beendet werden.

Router

```
router.all(path, [callback, ...] callback);
router.METHOD(path, [callback, ...] callback);
router.route(path).get(func).post(func);// etc
```

Path verwendet pattern-matching (/ab*c) und unterstützt **dynamische Werte** (/books/:id/) -> req.params.id.

View Engine (Default Jade)

```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'hbs'); // express-hbs
// Usage
app.render(view [, locals], callback);
res.render(view [, locals] [, callback])
```

Cookie/Session

```
app.use(require('cookie-parser')([secret][, options]));
// secret für Signierung
app.use(require('express-session')({ secret:
,'1234567', resave: false, saveUninitialized: false}));
```

jQuery Ajax

get(url [, data][, success][, type]), **getJSON**(...), **post**(...), **ajax**(url [, settings])
Promise: .done(), .fail(), .always(), .then()

SOP/CORS

Same Origin Policy: Protocol, Domain, Port
Cross-Origin Ressource Sharing: Ziel muss erlaubnis erteilen.

```
app.use(function(req, res, next) {
 res.header('Access-Control-Allow-Origin', '*'); //
Erlaube Anfrage von...
 res.header('Access-Control-Allow-Methods',
'GET,PUT,POST,DELETE');
 res.header('Access-Control-Allow-Headers', 'Content-
Type');
 next(); };
```

REST (Repres. State Transfer)

Eigenschaften: Trennung von Client- und Server-Logik; Stateless; Cache'bar (einzelne Antworten); Uniform Interface (Selbstbeschreibend); Erlaubt Schichtenarchitektur
Identifikation via URI; Ressource ist unabhängig der Repräsentation; Manipulation über Repräsentation

ROA (Ressource Oriented Arch.)

Alles was genug wichtig ist um eigenständig referenziert zu werden.
Name: z.B. URI; **Repräsentation**: z.B. JSON, XML; **Links**: Verknüpfung von Daten; **Interface**: Standard-Methoden; **Stateless**
URI: Ressourcen Mehrzahl, Query-Parameter nur für Algorithmen/Filter
GET: Read; **POST**: Erzeugt neue Res. und ID, Location-Header enthält neue URI, kein Cache; **PUT**: aktualisiert/erzeugt, idempotent, kein Cache; **DELETE**: löscht Ressource, kein Cache; **OPTIONS**: Beschreibt Methoden; **HEAD**: leifert nur Header der Res.; **PATCH**: partielles Update
-> Immer Statuscodes beachten.

HATEOAS

Hypermedia As The Engine Of ApplicationState -> Prozessgedanke in der Ressource (Navigation, Aktionen, Beschreibung, etc.)

SASS/CSS

& - Referenz zu parent // **@extend** smth – erben von
$ - Variable // **@mixin** smth(param) - mehrere Zeilen
%smth – Placeholder/Abstrakt // **@each** $i in $items
@function do($param){@return $param;}
@if <expr> {...} **@else** {...}

CSS Eigenschaft: box-sizing

content-box: Default, width & height betreffen nur den Content; **border-box**: Content, Border & Padding ohne Margin

Responsive Design

Viewport

Sklaieren des Viewports auf ideal viewport nötig

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

Flexbox (CSS)

```
.container{ display: flex; flex-direction: row[-
reverse]|column[-reverse]; flex-wrap: [no]wrap[-
reverse]; justify-content: flex(-start|-end)|center|
space(-between|-around); align-items: flex(-start|-
end )|center|baseline|stretch; align-content: flex(-
start|-end)|center|space(-between|-around)|stretch;}
.item{flex: none | [ <'flex-grow'> <'flex-shrink'>? ||
<'flex-basis'> ]; align-self: <s. Align-items>}
```

@Media Query

Screengrössen abfragen (Operatoren and, not, only):

```
@media only screen and (min-device-width: 123px) and
(max-device-width: 667px) and (-webkit-min-device-
pixel-ratio: 2) {…}
```

Responsive Bilder

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

srcset: Liste mit Bilder mit Grösse(w) oder Pixeldichte (x)
sizes: Media-Query mit Grösse für Bild im Layout

Responsive Layout Pattern

M:Mobile; T:Tablet; N:Notebook; D:Desktop
Mostly Fluid: M:Unterein., TND: Titel oben, Rest unten nebene.
Column Drop: MT:Unterein., N:Menü, D:Nebeneinander
Layout Shifter: MT:Untere.., ND:Menü, sonst untere.
Reflow: M: Eine Spalte, TND: Viele Spalten +Bilder +Zeilenlänge + Platznutzen
Expand: Max Breite, TND: mit Rand +Bilder +Zeilenl.
Sidebar für Landscape: +Bildschhöhe +Bild +Zeilenl.
Master d. Detail: +Platzn. +Kontext +Zeilenl.
Off Canvas: Ausblenden von Menü
Off/On-Screen Menü: +Platzn. +Kontext

Formular

```
.container{ display: flex; flex-wrap: wrap; align-items: center; }
.label { flex: 1 0 120px; max-width: 220px; /* GSB*/
.input { flex: 1 0 220px; }
```

User Centered Design

Garett Ebenen

Oberfläche: Sensorisches Design
Raster: Informationsdesign, Interfacedesign, Nav.d.
Struktur: Interaktionsd., Informationsarch.
Umfang: Funktionsspez., Inhaltsanforderungen
Strategie: Nutzerbedürfn., Produktziele

Interaktion nach Norman

Nutzer –(Visibility, Affordance(Begreifbark.))→ Anwendung [Kluft d. Ausführung]
Anwendung -(Feedback)→ Nutzer [Kluft d. Evaluation]

JS OO

```
// 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();
//Constructor
const Counter=function(initDelta, initCount){ let delta
= initDelta||1; this.count = initCount||0; this.countUp
= function(){this.count+=delta};};
// binding „this“ to scope
const cnt=Counter(9);
const incFunc=cnt.countUp.bind(cnt); // Definiert this
```

ES6 Sprachfeatures

Default-Werte für Parameter, falls Argument undefined; **Rest-Parameter** (Spread Operator); **Objekt-Parameter** erlauben "named Arguments", benötigen Default; **const/let** anstatt var, Runtime-Type-Errors; **Temporal Dead Zone**, Variablenzugriff vor Deklaration Runtime-Reference-Error; Implizite **Property-Names**, **Template-Strings** mit ${exp} und Backtick; Property **Getters/Setters**; **Methods** direkt in Objekt; Array-funktionen behalten this; **Class**-Keyword inklusive **Constuctor**-Method -> super()

```
function car({brand='VW'}={}, ...pass){…}
var str1 = `Result: ${a + b}`;
```

Modularisierung

Revealing Module Pattern

Generierung innerhalb eines iife, export-Objekt wird globaler Variable zugewiesen. (Bsp: jQuery)

```
(function(){window.Counter = C;})(); // iife
```

Konfliktlösung durch Package-Objekte.

Require.JS / Async. Module Def.

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

Modul explizit laden (eager loading):

```
require(['./Counter'], function(Counter){…});
```

Definition des Modules:

```
define([(<requiredModuleString>,)*],
<functionReturningExportObject>
```

Module werden bei Bedarf nachgeladen, auch nach Aufbau des DOM-Trees.

ES6 import/export

```
import Counter from './Counter';
export default Counter; // Counter.js
```

Nicht von Browsern unterstützt, ES6 Präprozessor nötig (z.B. Babel).

Testing

Depend-on Component (DOC), System Under Test (SUT)

Unit Test Patterns

Test Double: Ersetzt depend-on component; entfernt Abhängigkeiten; Verhaltet sich gleich
Mock Object: Verifiziert indirekte Outputs des SUT

Jasemine Test Specification

Test Suite (Test Fixture) > Specs (Test Cases) > Expectations (Assertions)

```
describe("Suite", function() { //Variables, …
 beforeEach(function() {…});
 afterEach(function() {…});
 it("fulfills specification", function() {
   var foo = 1
   expect(foo).toEqual(1); // Matcher
 });
});
```

Unit Test Smells

- **Test Code Duplication** -> Extract Method
- **Conditional Test Logic** -> Test Double
- **Obscure Test:** Absicht unklar -> Single condition tests
- **Test Logic in Production** -> Test Double
- **Assertion Roulette** -> Single condition tests
- **Slow Tests**: Schnittstellen zu Umsystemen; grosse Fixtures -> Test Double; Minimum Fixtures