



Nodejs Grundlagen

Michael Reichart, Dezember 2022

2021 Michael Reichart, copyleft CC BY-NC-SA

GFU CYRUS AG, KÖLN



- IT Fortbildung für Unternehmen
- 180+ erfahrene Dozenten, die wissen, was sie tun.
- 1800+ Seminare aus den Bereichen Anwendung, Administration, Entwicklung, Organisation.
- Alle Schulungen können präsent, **remote** oder hybrid durchgeführt werden.
- Ein routiniertes, serviceorientiertes, begeistertes Mitarbeiterteam.
- Die beste Candybar!

FORTBILDUNG ENTLANG DES SOFTWARE LIFECYCLE



MICHAEL REICHART

DESIGNER, CODER, COACH

- **Gestalter** für visuelle Kommunikation
- **Softwareentwickler** für Browser-/Server-Software
- 1996 - 2017 CEO der Digitalwerkstatt Stuttgart
- Seit 2016
Lead Trainer der GFU Cyrus AG,
Blended Learning Consultant und Lerntransfer Designer,
- Seit über 20 Jahren Dozent und Coach
für Unternehmen und Hochschulen, Trainer der GFU Cyrus AG.
Seit 8 Jahren Wahl-Kölner



SEMINARE FÜR ...

- HTML5, CSS und Javascript (**nodejs**)
- Physical Computing (Microcontroller, Elektronik, Software)
- Datenvisualisierung und Informationsgrafik
- Softwarearchitektur
- UX/UI (Konzeption, Herangehensweisen)
- Barrierefreiheit (Web-Apps, Word, PDF)



- Auf LinkedIn unter Michael Reichart.
- Per Mail: michael.reichart@gfu.net



9:00 Uhr bis 16:00 Uhr
Mittag 12:00 Uhr – 13:00 Uhr

kollegial?



formell?





Ziele?

Sicherheitsrelevante Services auf den Server verlagern

D3

DU

Synchron/Asynchron

Visualisieren von (Laufzeit-) Daten

MySQL-Anfragen (asynchron)

callback-hell

Modularer Auf-/ Umbau eines Monolithen

Wer bist Du und was ist Deine Aufgabe im Unternehmen?

Wie schätzt Du Dein bisheriges Verständnis für nodejs/Javascript ein?

Nichts

Ich habe **bereits Erfahrungen**, aber das reicht nicht.

Ich glaube, ich weiss schon alles!

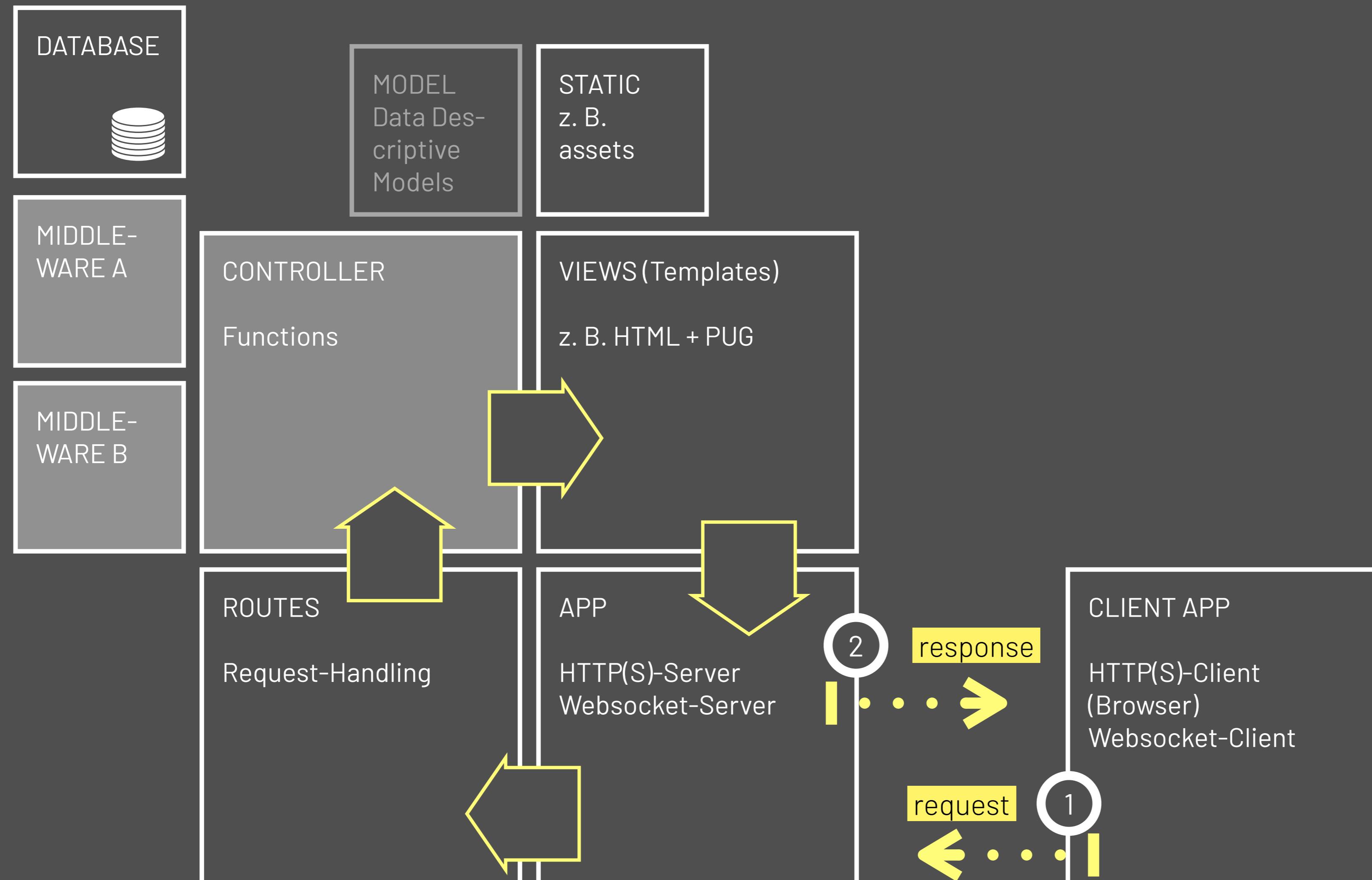
Ich suche eine Stelle als IT Trainer.

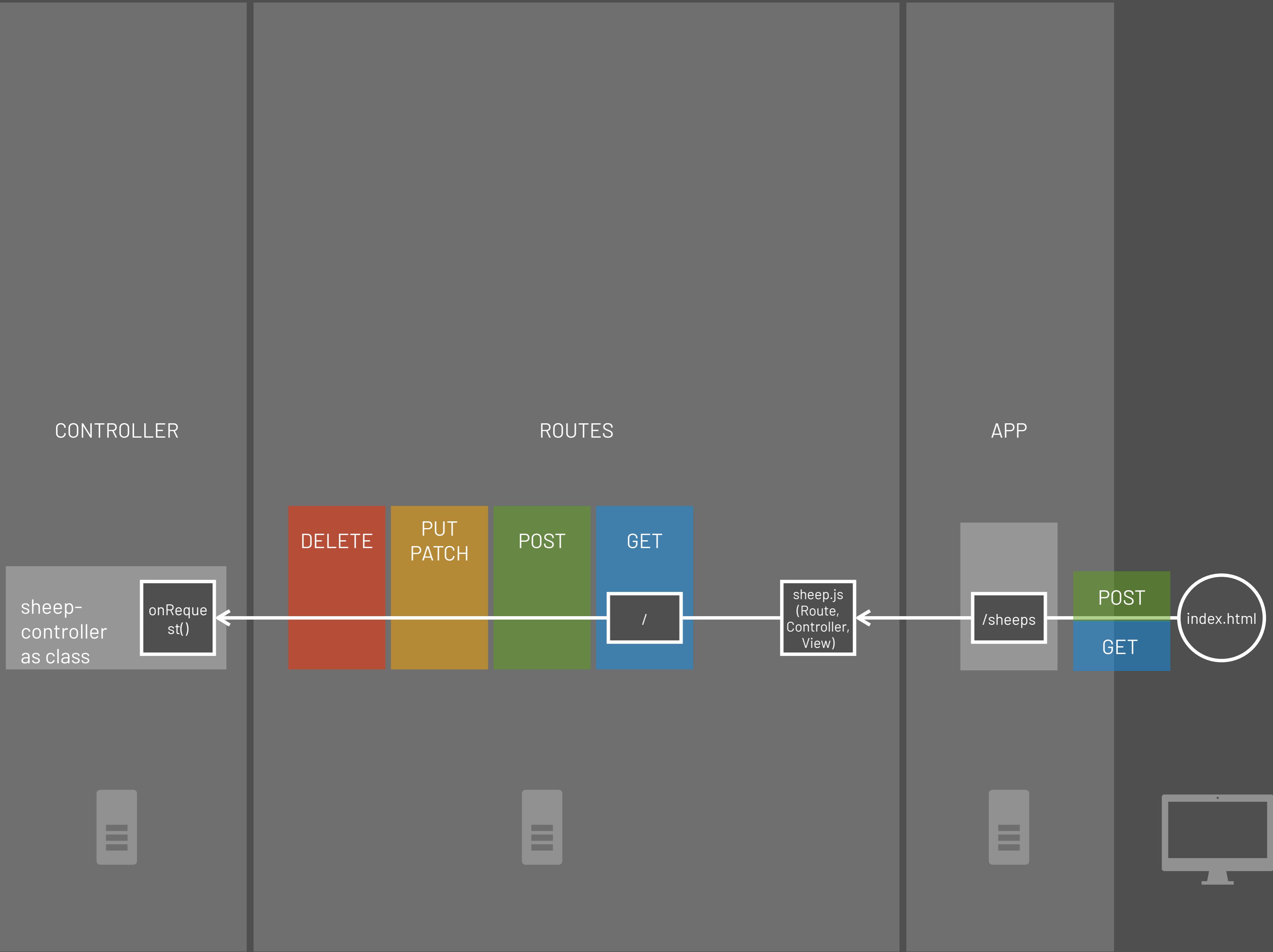
Sehr wenig, ich bin definitiv **Anfänger**.

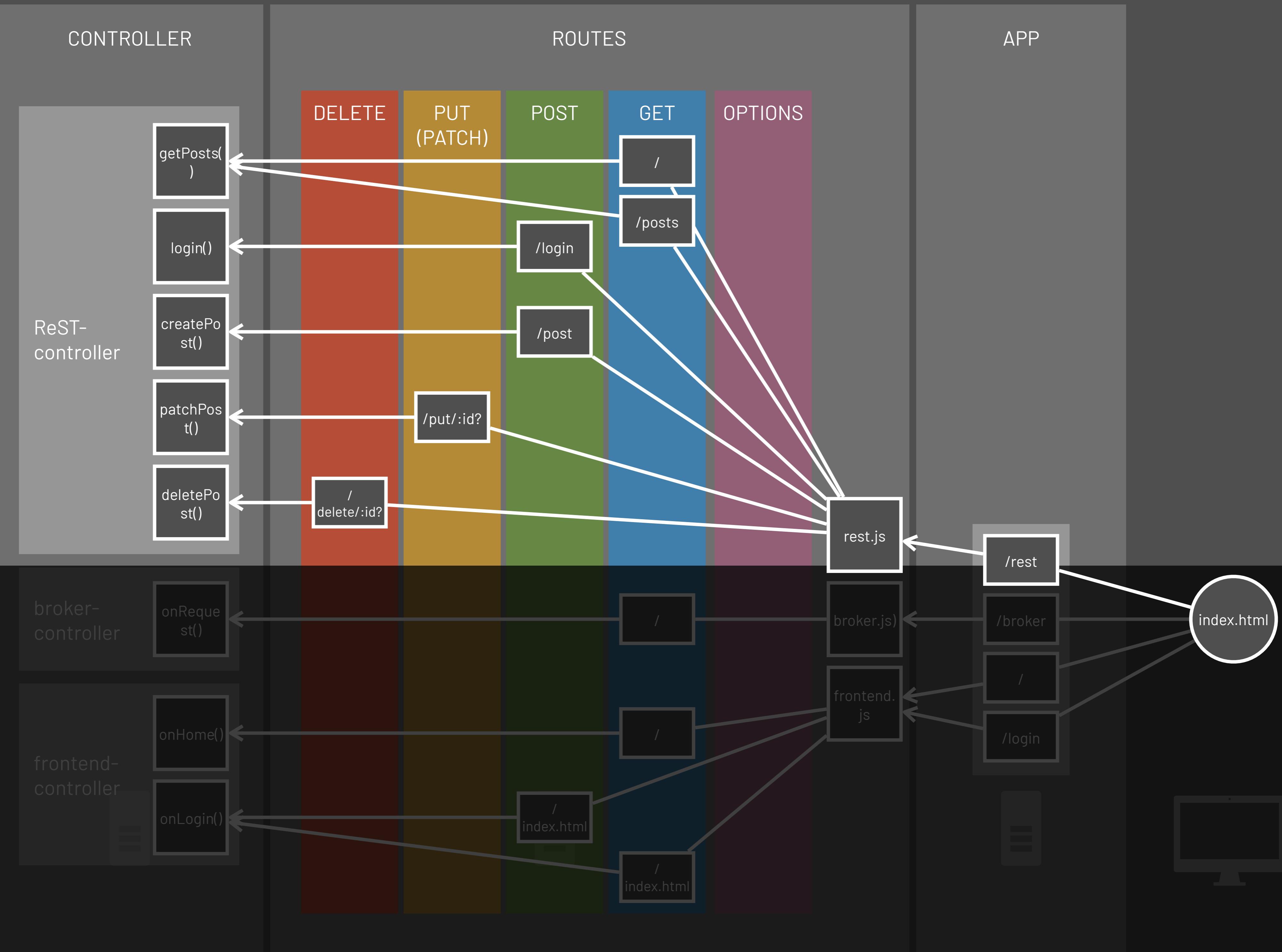
Ich kenne mich schon gut aus, möchte aber **besser verstehen**.

viel Erfahrung









Einführung in node.js

- + Über den Autor Ryan Dahl, die Javascript Engine V8 und die Idee für serverseitiges Javascript

„It will be very fun.“

-Ryan Dahl

Node.js was created by Ryan Dahl starting in 2009. Its development and maintenance is sponsored by Joyent.

-Wikipedia

Ryan Dahl, the creator of Node.js, originally had the goal in mind of creating web sites with push capabilities as seen in web applications like Gmail. After trying solutions in several other programming languages he chose JavaScript because of the lack of an existing I/O API. This allowed him to define a convention of non-blocking, event-driven I/O.

-Wikipedia

„Node.js ist eine serverseitige Plattform zum Betrieb von Netzwerkanwendungen. Insbesondere lassen sich Webserver damit realisieren. Node.js basiert auf der JavaScript-Laufzeitumgebung „V8“, die ursprünglich für den Chrome-Browser entwickelt wurde und bietet daher eine ressourcensparende Architektur, die eine besonders große Anzahl gleichzeitig bestehender Netzwerkverbindungen ermöglicht.“

-Wikipedia

Node.js was first published by Dahl in 2011 and could only run on Linux. NPM, a package manager for Node.js libraries, was introduced the same year.

In June 2011, Microsoft partnered with Joyent to help create a native Windows version of Node.js. The first Node.js build to support Windows was released in July.

-Wikipedia

On January 30, 2012 Dahl stepped aside, promoting coworker and NPM creator Isaac Schlueter to the gatekeeper position.

...

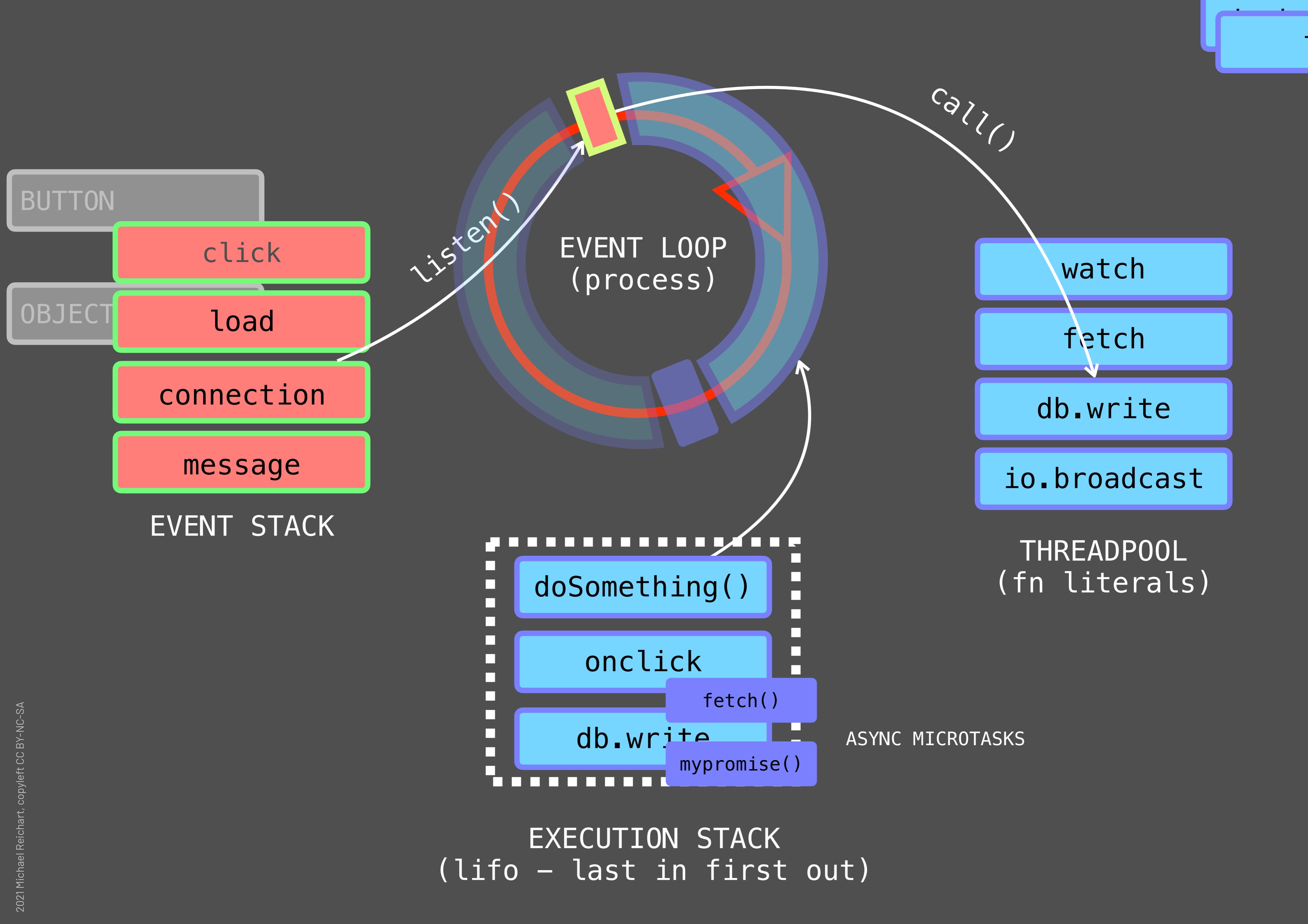
On January 15, 2014 Schlueter announced he was making NPM his main focus and Timothy J Fontaine would be Node.js new project lead.

-Wikipedia

JavaScript auf dem Server zu verwenden, ist gut, weil

- + Client und Server technisch auf derselben Basis stehen.
- + JavaScript alle Aufgaben eines Web-, Socket- oder Datenservers übernehmen kann.
- + Format- und Kompatibilitätsprobleme zwischen unterschiedlichen Sprachen, zum Beispiel Javascript und PHP wegfallen.
- + Server und Client zu einem Ganzen verschmelzen.

Execution Stack, Threads and Eventloop



Installation und Konfiguration



[HOME](#) | [ABOUT](#) | [DOWNLOADS](#) | [DOCS](#) | [FOUNDATION](#) | [GET INVOLVED](#) | [SECURITY](#) | [NEWS](#)

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.

16.18.1 LTS

Recommended For Most Users

19.10.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [LTS schedule](#).

 **LINUX FOUNDATION** COLLABORATIVE PROJECTS

[Report Node.js issue](#) | [Report website issue](#) | [Get Help](#)

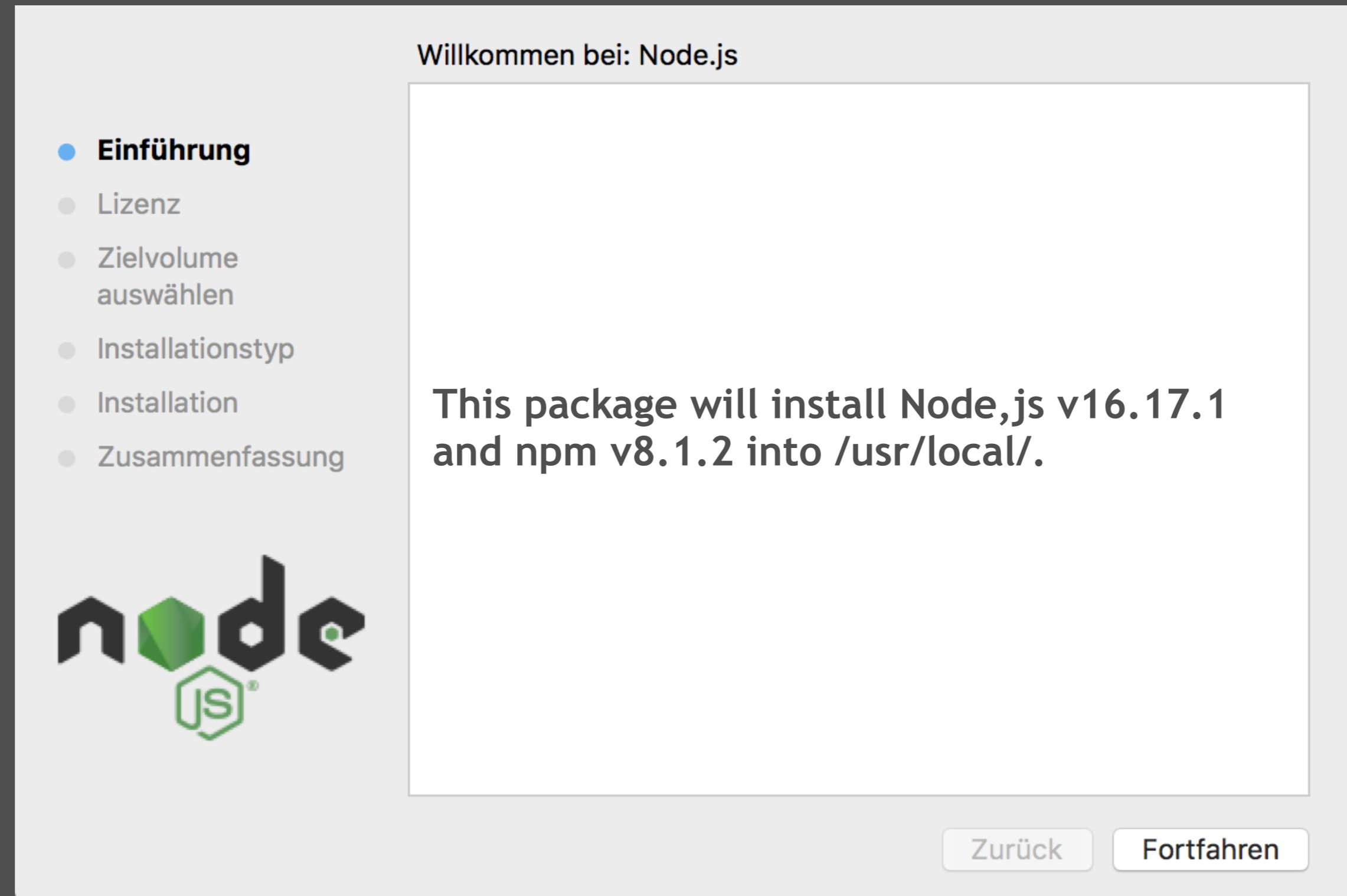
© 2017 Node.js Foundation. All Rights Reserved. Portions of this site originally © 2017 Joyent.

Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the Trademark Guidelines of the Node.js Foundation.

Linux Foundation is a registered trademark of The Linux Foundation.

Linux is a registered trademark of Linus Torvalds.

- + Installer für Windows und Mac OS
- + Erweiterbar über Node Package Manager
- + geschrieben in C++, individuell kompilierbar



- Einführung
- Lizenz
- Zielvolume auswählen
- Installationstyp
- Installation
- **Zusammenfassung**



Installation erfolgreich abgeschlossen

Node.js was installed at

`/usr/local/bin/node`

npm was installed at

`/usr/local/bin/npm`

Make sure that `/usr/local/bin` is in your \$PATH.

[Zurück](#)

[Schließen](#)

Node abfragen

```
$ node -v  
v16.3.0
```

```
$ npm -v  
v7.15.1
```

Nodejs via Terminal aktualisieren

```
[\$ npm cache clean -f]  
[\$ npm install -g n]
```

```
npm install -g npm
```

```
$ n stable
```

```
$ n latest
```

```
$ n 7.8.0
```



Search packages

Search

Sign Up

Sign In

Build amazing things

We're npm, Inc., the company behind Node package manager, the npm Registry, and npm CLI. We offer those to the community for free, but our day job is building and selling useful tools for developers like you.

Take your JavaScript development up a notch

Get started today for free, or step up to npm Pro to enjoy a premium JavaScript development experience, with features like private packages.

🔥 Popular libraries

lodash

react

chalk

tslib

axios

express

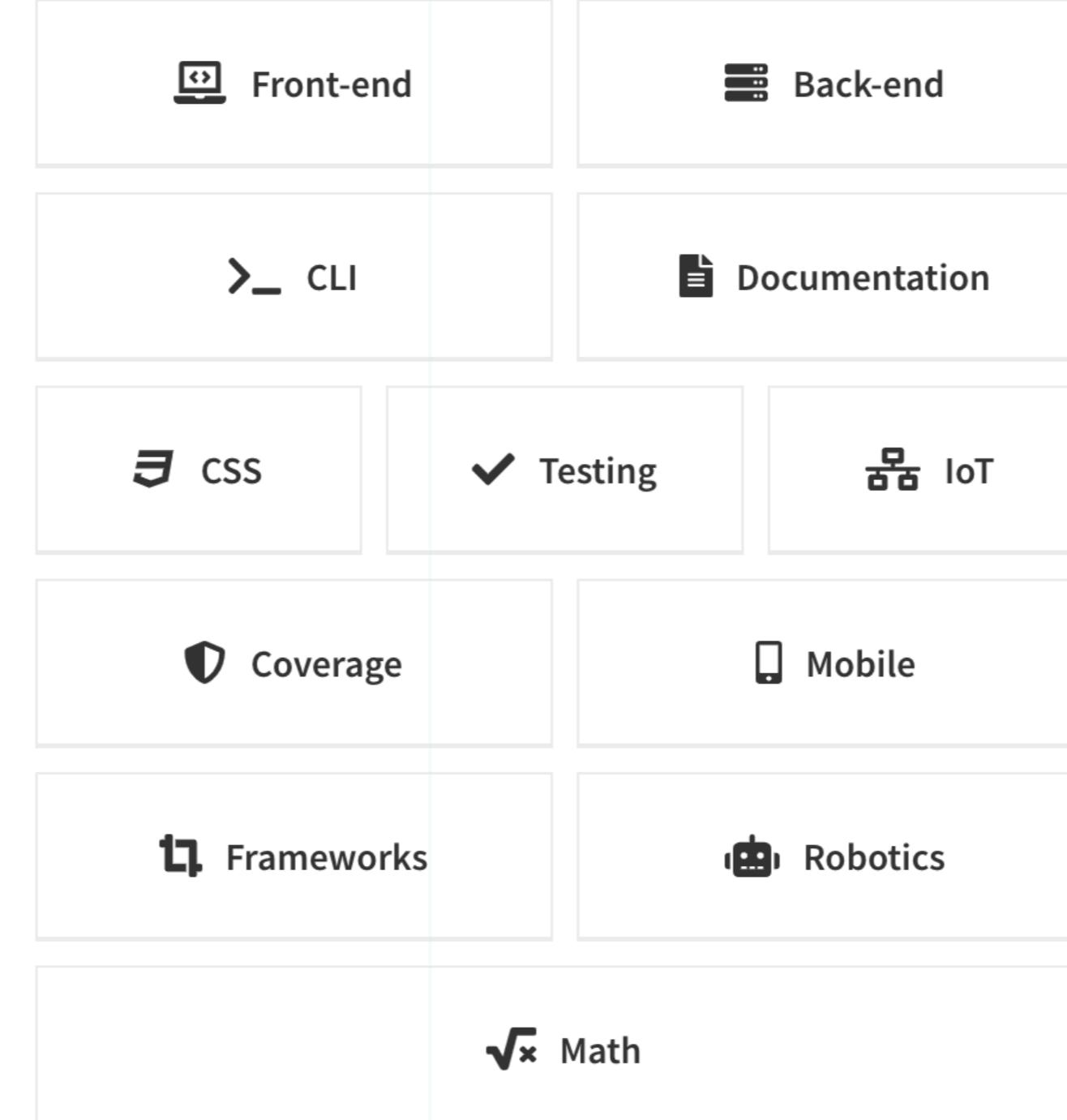
commander

moment

request

react-dom

🔍 Discover packages



📊 By the numbers

Packages

1.639.776

Downloads · Last Week

31.490.008.779

Downloads · Last Month

126.857.612.674

Hallo Welt
4 Versionen

Hallo Welt - Konsolenversion

```
$ node
> function hw () { console.log("hello world"); }
> hw()
hello world
```

(To exit, press ^C again or type .exit)

Hallo Welt - Programmversion

```
// hello world.js
// -----
function hw () { console.log("hello world"); }
hw();

// -----
$ node hello-world
hello world
$
```

server.js

```
import http from 'http';          // aka. http = require('http')

let host = 'http://localhost',
    port = 3000,
    server = null;

server = http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.end('<h1>Hello world</h1>');
});

server.listen(port);

console.log('a simple web service on ' + host + ':' +
port);
```

server.js ausführen

```
$ node server
```

app.js (Express-Version)

```
// Modules
// const path = require('path');
// const express = require('express');
// const pageProcessor = require('./routes/pageProcessor');

// ES Modules
import path from 'path';
import express from 'express';
import pageRoute from './route.js';

const app = express();

let port = 3000,
    host = 'localhost';

app.use(express.static(path.resolve('public')));

app.set('view engine', 'ejs');
app.set('views', path.resolve('./'));

app.use('/', pageRoute);

app.listen(port, () => console.log(`Server running at http://${host}:${port}`));
```

Projektstruktur

```
app.js  
route.js  
view.ejs
```

```
data.json
```

```
/static  
  /assets  
    /css  
      style.css  
  /js  
    script.js
```

view.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width">
  <title>Table View</title>
  <link rel="stylesheet" href="assets/css/style.css">
</head>

<body>
  <h1>Page View</h1>
  <p><%= data %></p>

  <script src="assets/js/script.js"></script>
</body>

</html>
```

route.js

```
import express from 'express';
import path from 'path';

const router = express.Router();

import {
  readFile
} from 'fs/promises';

const data =
  JSON.parse(
    await readFile(
      new URL('./data.json',
        import.meta.url)
    )
  );

function onRequest(request, response) {
  response.render('view', {
    data: data.message
  });
}

router.get('/', onRequest);

export default router;
```

data.json

```
{  
  "message": "hello World",  
}
```

Um Ports unterhalb der 1024 zu verwenden, muss das Node.js-Programm mit Root-Rechten ausgeführt werden.

-Gut zu wissen.

Node Monitoring



nodemon reload, automatically.

Nodemon is a utility that will monitor for any changes in your source and automatically restart your server. Perfect for development. Install it using [**npm**](#).

Just use **nodemon** instead of **node** to run your code, and now your process will automatically restart when your code changes. To install, get [**node.js**](#), then from your terminal run:

```
npm install -g nodemon
```

Features

- Automatic restarting of application.
- Detects default file extension to monitor.
- Default support for node & coffeescript, but easy to run any executable (such as python, make, etc).

Nodemon global installieren

```
$ [sudo] npm install -g nodemon
Password:

.
.

+ nodemon@1.18.9
updated 1 package in 7.654s

.
.

npm config -g set proxy http://172.16.20.127:8080/
npm config -g set https-proxy http://172.16.20.127:8080/
```

Auto restart mit nodemon

```
$ nodemon hello-world.js  
$ nodemon --debug ./server.js 80
```

Nodemon konfigurieren

```
{  
  "name": "nodemon",  
  "homepage": "http://nodemon.io",  
  ...  
  ... other standard package.json values  
  ...  
  "nodemonConfig": {  
    "ignore": ["test/*", "docs/*"],  
    "delay": "2500"  
  }  
}
```

About Nodemon

<https://github.com/remy/nodemon>

Ein Node Projekt aufsetzen

Ein Nodeprojekt initialisieren

```
npm init
```

npm fragt alle Informationen für das Projekt ab und erzeugt dann eine package.json.

package.json Beispiel

```
{
  "name"      : "my-project",
  "version"    : "1.0.0",
  "description": "Lorem descriptum ipsum dolor sit.",
  "main"       : "./index.js",
  "directories": {
    "doc": "docs"
  },
  "repository": {
    "type"  : "git",
    "url"   : "git+https://github.com/account/
               project.git"
  },
  "keywords"  : [ "comma", "separated",
                 "value", "and", "word", "list" ],
  "author"    : "Michael Reichart",
  "license"   : "MIT",
  "bugs": {
    "url" : "https://github.com/account/
             project/issues"
  },
  "homepage" : "https://github.com/account/
                project#readme",
  "type"      : "module",
  "scripts"   : {
    "watch" : "nodemon",
    "test"  : "echo \"Error: no tests\" && exit 1"
  },
  "dependencies": {
    "ejs"    : "^3.1.7",
    "express": "^4.18.1",
  },
  "devDependencies": {
    "connect-livereload": "^0.6.1",
  }
}
```

Ein Nodeprojekt faul initialisieren

```
npm init -y
```

npm schreibt alle hinterlegten Informationen automatisch in eine package.json.

Ein Nodeprojekt initialisieren

npx license mit

uses the license package to download a license of choice, in this case the MIT license

npx gitignore

node uses the gitignore package to automatically download the relevant .gitignore file from GitHub's repo

npx covgen

uses the covgen package to generate the Contributor Covenant and give your project a code of conduct that will be welcoming to all contributors

Initiale npm package-Werte einstellen

```
npm config list | grep init

npm config set init.author.name "Your name"
npm config set init.author.email "your@email.com"
npm config set init.author.url "https://your-url.com"
npm config set init.license "MIT"
npm config set init.version "1.0.0"
```

Einmal hinterlegt, werden bei `npm init -y` die richtigen Werte eingesetzt.

Ausgaben über die Konsole

Ausführungszeit messen

```
console.time('my time measuring');

let value = 'a primitive value',
    object = {
        prename : 'Michael',
        lastname : 'Reichart'
    };

console.log(value);
console.dir(object);

console.timeEnd('my time measuring')
```

Konsolenbefehle

```
console.assert(value[, message][, ...args])  
console.dir(obj[, options])  
console.table(obj)  
console.error([data][, ...args])  
console.info([data][, ...args])  
console.log([data][, ...args])  
console.time(label)  
console.timeEnd(label)  
console.trace(message[, ...args])  
console.warn([data][, ...args])
```

Debugging

Manual Debugging

```
$ node inspect hello-world.js

< Debugger listening on 127.0.0.1:5858
connecting to 127.0.0.1:5858 ... ok
break in hello-world.js:2
1 // -----
> 2 console.time('processed time');
3 // -----
4
debug>
```

server.js

```
import http from 'http';

let host = 'http://localhost',
    port = 3000,
    server = null;

server = http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    debugger;
    response.end('<h1>Hello world</h1>');
});

server.listen(port);

console.log('a simple web service on ' + host + ':' +
port);
```

Stepping

`cont, c` - Continue execution
`next, n` - Step next
`step, s` - Step in
`out, o` - Step out
`pause` - Pause running code

Breakpoints

```
setBreakpoint(),  
sb()
```

– Set breakpoint on current line

```
setBreakpoint(line),  
sb(line)
```

– Set breakpoint on specific line

```
setBreakpoint('fn()'),  
sb(...)
```

– Set breakpoint on a first statement
in functions body

```
setBreakpoint('script.js', 1),  
sb(...)
```

– Set breakpoint on first line of

script.js

```
clearBreakpoint('index.js', 15),  
cb(...)  
15
```

– Clear breakpoint in index.js on line

Debugging mit den Chrome Devtools

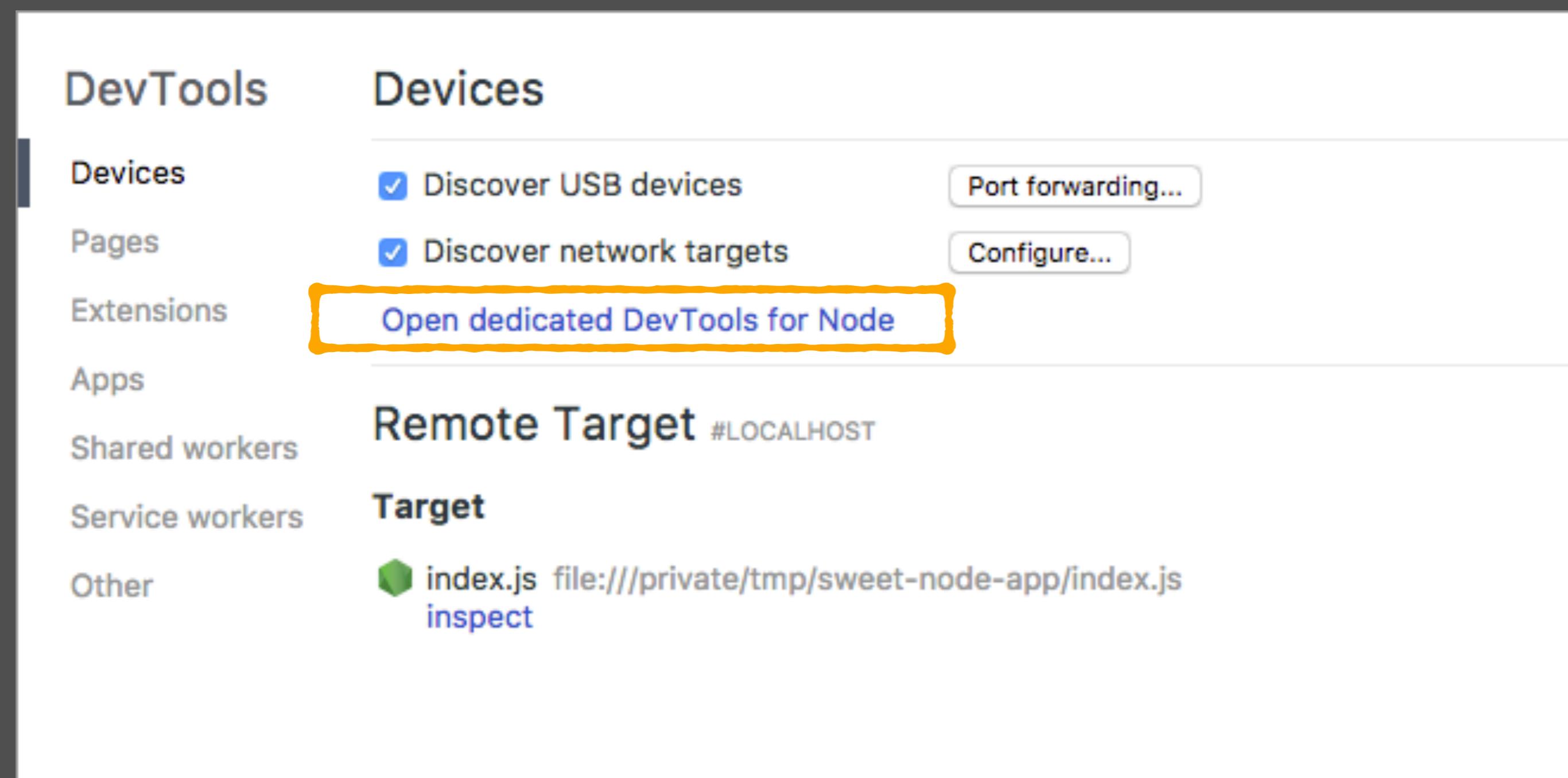
Der Nodejs V8 Inspector

- + Die Chrome Extension "**nodejs V8 inspector**" in einer aktuellen Chromeversion installieren.
- + startet einen Port 9229, über den die Chrome Devtools Fehlermeldungen anzeigen und der Code fehlerbehandelt werden kann.
- + `node --inspect index.js`

`# Break on the first statement of the script with --inspect-brk`

`node --inspect-brk index.js`

- <chrome://inspect>



Software Architektur - Sichten und Schichten

- + Anwendungsarchitekturen
(Softwarearchitektur)
OSI, 4+1 Sichten, Client-
Server, 3- und N-Tier-
Systeme

4+1 - Sichten-Modell

- + **logical view** - Funktionalität des Systems für den Endnutzer. (Lastenheft-Team Fragen den Endnutzer)
- + **process view** - sie verdeutlicht die Prozesse des Systems, sowie deren Laufzeitverhalten und Kommunikation. Die Prozessicht beschreibt Parallelität, Verteilung, Integration, Performance und Skalierbarkeit. (Requirements Engineer)
- + **development view** (implementation view)
- beschreibt das System vom Standpunkt eines Entwicklers. (20% Scrummaster als Teamkoordinator)
- + **physical view** (deployment view)
beschreibt das System vom Standpunkt des Systemarchitekten. Es beschäftigt sich mit der Verteilung der Softwarekomponenten auf physikalischer Ebene (Hardware) und der Kommunikation zwischen diesen Komponenten.
- + **scenarios**: Die fünfte Sicht zeigt wichtige Anwendungsfälle oder Anwendungsszenarien. Sie beschreiben Abläufe zwischen Komponenten bzw. Prozessen. Sie helfen, Architekturelemente zu identifizieren, zu veranschaulichen und die Architektur zu überprüfen. Sie dienen auch als Startpunkt für erste Architekturtests bzw. Implementierungsentwürfe.

Sicht des Anwenders

UML-Diagrammformen:

Klassendiagramm,
Kommunikationsdiagramm,
Sequenzdiagramm
u.a.

Dynamik des Systems;
Laufzeitverhalten

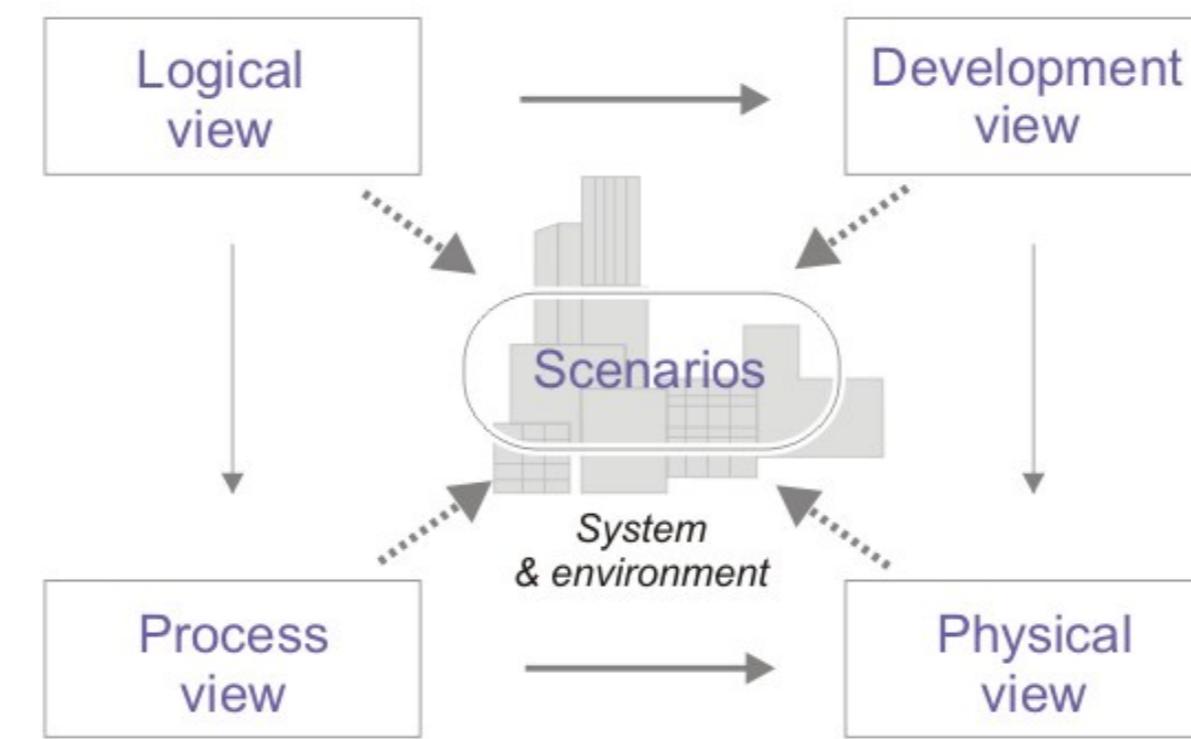
UML-Diagrammform:
Anwendungsfalldiagramm

Sicht des Entwicklers

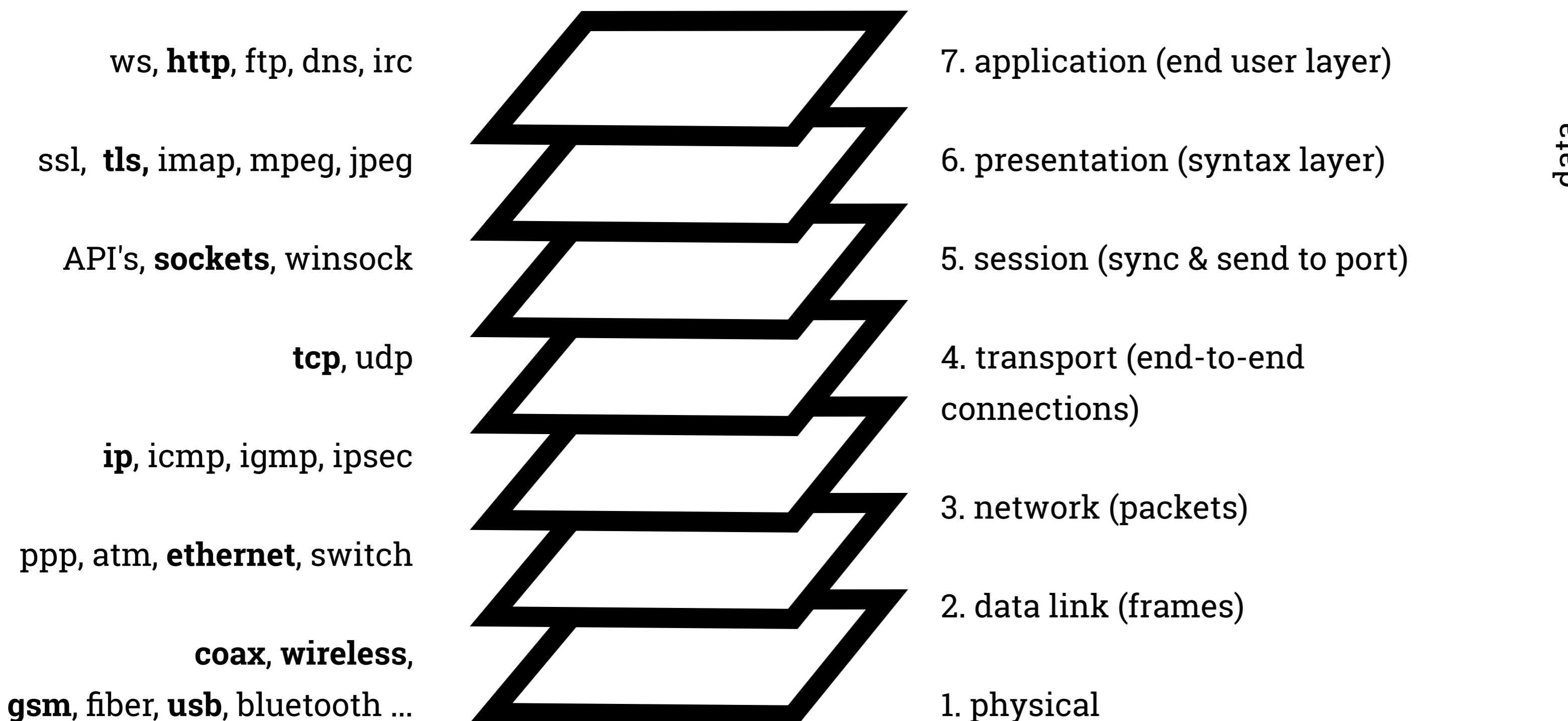
UML-Diagrammformen:
Komponentendiagramm,
Paketdiagramm

Sicht des
Systemarchitekten

UML-Diagrammform:
Verteilungsdiagramm



OSI - Open Systems Interconnection



Die sieben Ebenen des OSI Modells

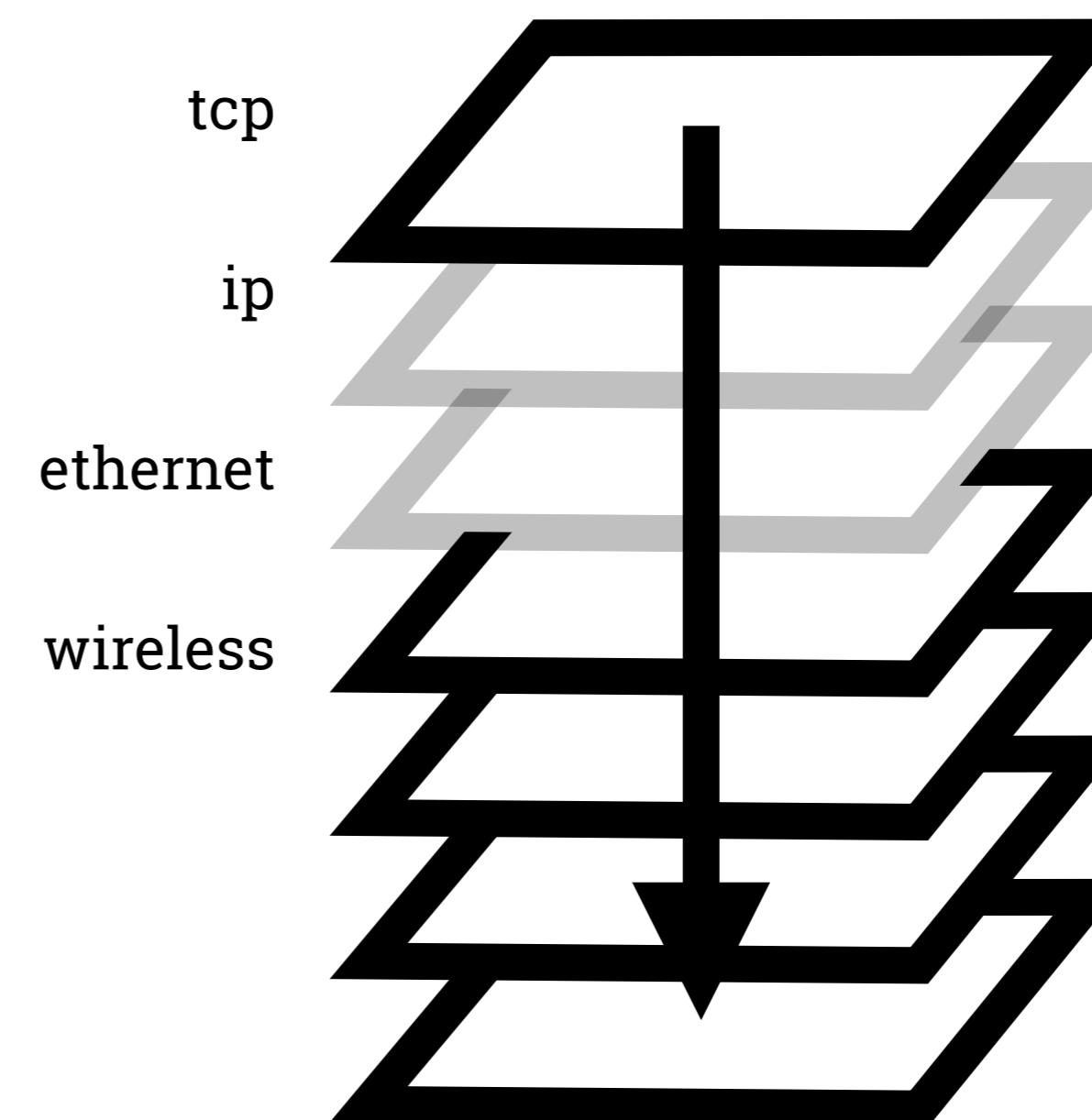
OSI - Open Systems Interconnection

If you use your web browser to navigate to <http://www.gfu.net>, this communication uses the following protocols from each layer, starting at layer 7:

HTTP → TCP → IP → Ethernet.

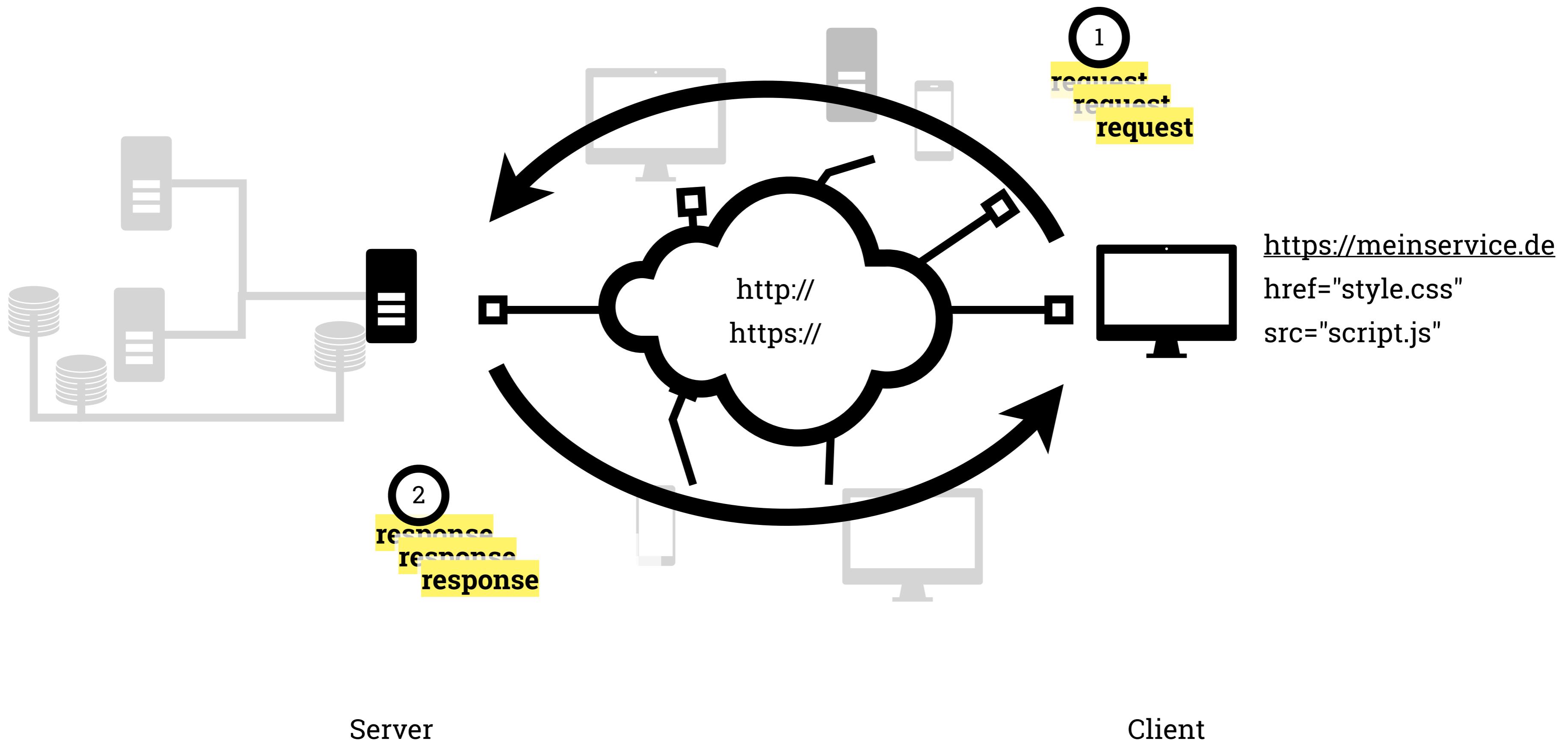
On the other hand, entering <https://www.gfu.net> would use

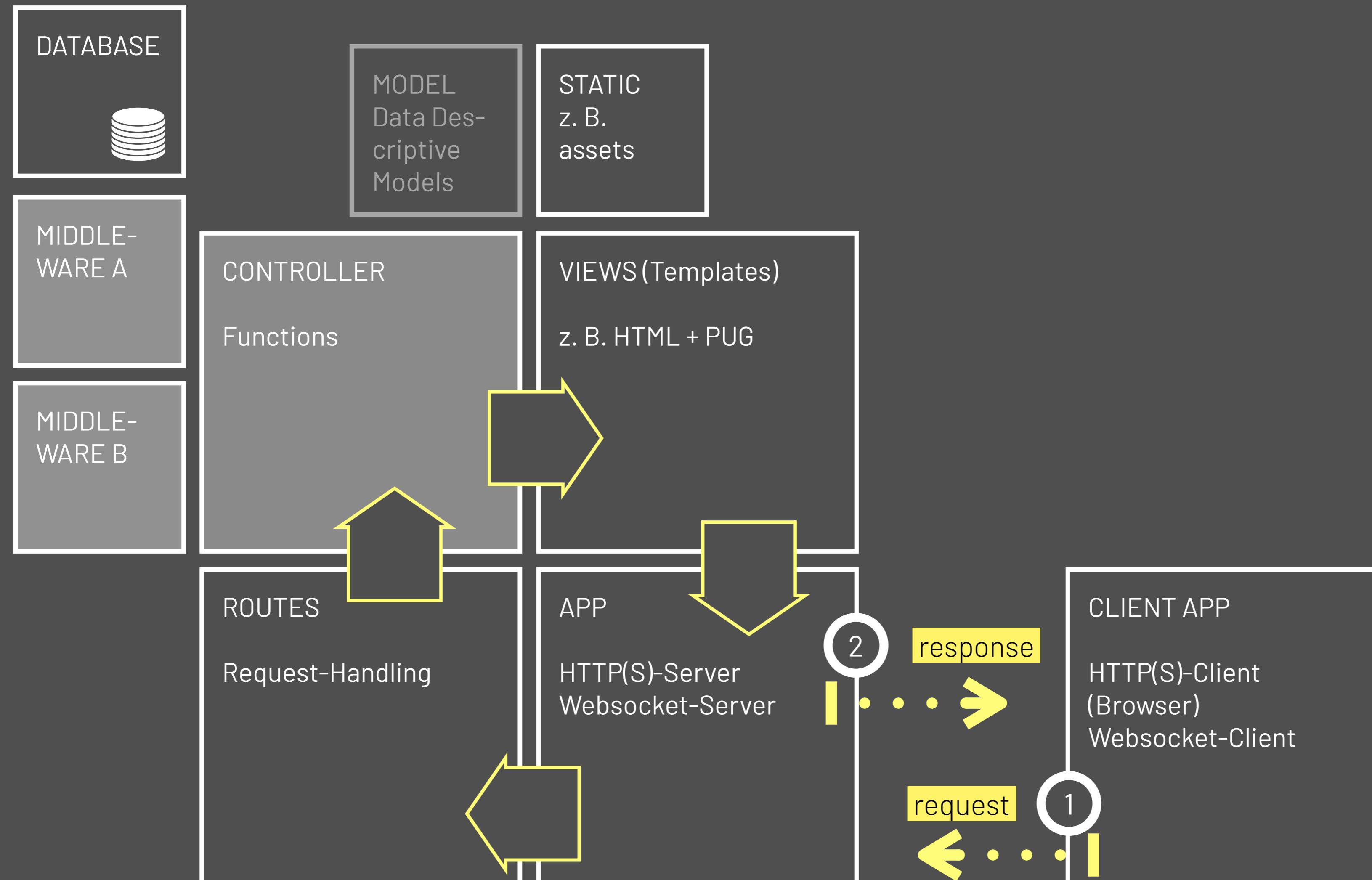
HTTP → SSL → TCP → IP → Ethernet.



7. application (end user layer)
6. presentation (syntax layer)
5. session (sync & send to port)
4. transport (end-to-end connections)
3. network (packets)
2. data link (frames)
1. physical

Server - Netzwerk - Client





Ein nodejs-Modul aufbauen

ECMAScript Module vs. Common Modules

- + EScript Module sind der offizielle Standard, um wiederverwendbaren Code zu schreiben.
- + ES Modules verwenden **import** and **export** Statements.
- + ES Module verwenden die Dateiendung **.js** oder **.mjs**.
- + CommonJS Module folgen einer nodejs-Syntax. Jede nodejs-Datei bildet automatisch ein eigenes Modul.
- + CommonJS Module verwenden **require()** und **export** Statements.
- + ES Module verwenden die Dateiendung **.js** oder **.cjs**.

Ein CommonJs Modul schreiben

```
// circle.cjs                                // app.js
let _area = null;
_area = function (radius) {
    return Math.PI * radius *
radius;
};

module.exports.area = _area;

const circle =
    require('./circle.cjs');
let radius = 5;
console.log(circle.area(radius));
> 78.53981633974483
```

Ein ES Modul schreiben und importieren

```
// circle.mjs                                // app.js

let area = function(radius) {                import {area} from './circle.mjs';
    return                                     let radius = 5;
        Math.PI * radius                      console.log( area(radius) );
    * radius;                                 > 78.53981633974483circle
};

// Named export
export {area}
```

```
// Benannter Export
function cube () { return value }
const foo = Math.sqrt(2);
export {cube, foo}

import {cube, foo} from ...

// default exportiert nur eines je Script
export default function() {}
export default class {}

import my-function from 'my-module'
```

Module verwalten und installieren

Abhängigkeiten in der package.json

```
"dependencies" : {  
    "express" : "^4.18.1",  
},  
  
"devDependencies" : {  
    "webpack"      : "*",  
    "webpack-cli" : "^4.9.2"  
}
```

Ein Module installieren

```
$ npm install
```

```
$ npm install express
```

```
$ npm install --save-dev webpack webpack-cli
```

Ein Modul einbinden

```
import http      from 'http';
import fs        from 'fs';
import express   from 'express';

// let
// http = require('http'),
// fs = require(fs),
// express = require('express');
```

Versionierung der Module

major.minor.patch

"modulename" : " v1.23.3"

Auf Grundlage einer Versionsnummer von MAJOR.MINOR.PATCH werden die einzelnen Elemente folgendermaßen erhöht:

MAJOR wird erhöht, wenn API-inkompatible Änderungen veröffentlicht werden.

MINOR wird erhöht, wenn neue Funktionalitäten, welche kompatibel zur bisherigen API sind, veröffentlicht werden.

PATCH wird erhöht, wenn die Änderungen ausschließlich API-kompatible Bugfixes umfassen.

Außerdem sind Bezeichner für Vorveröffentlichungen und Build-Metadaten als Erweiterungen zum MAJOR.MINOR.PATCH Format verfügbar.

Modul-Versionierung

```
{  
  "dependencies" : {  
    "modulename" : " v1.23.3",  
    "modulename" : " 1.23.3",  
  
    "modulename" : " >1.23.1",  
    "modulename" : ">=1.23.2",  
    "modulename" : "<=1.23.3",  
    "modulename" : " <1.23.4",  
  
    "modulename" : " >1.23.1 <1.23.4",  
    "modulename" : " 1.22.9 || >1.23.1 <1.23.4",  
  }  
}
```

Modul-Versionierung

```
"dependencies" : {  
    "modulename" : "1.23.1 - 1.23.5", // inclusive set  
    "modulename" : "1.2.3",           // === 1.2.3  
  
    "modulename" : "1.x",  
    "modulename" : "1.2.*",          // 1.2.15      -> 1.2.11 ist möglich  
  
    "modulename" : "~1",             // >= 1.0.0     < 2.0.0  
    "modulename" : "~1.2",           // >= 1.2.0     < 1.3.0  
    "modulename" : "~1.2.3",         // >= 1.2.3     < 1.3.0  
  
    "modulename" : "^1.2.3",          // >= 1.2.3     < 2.0.0  
    "modulename" : "^1.2.x",          // >= 1.2.0     < 2.0.0  
}
```

Wann wird die Versionsnummer angepasst?

- + Die Version **1.0.0** definiert die öffentliche API. Ab dieser Veröffentlichung hängt die Art und Weise, wie die Versionsnummer erhöht und verändert wird, von den folgenden Regeln ab.
- + Die Patch Version Z ($x.y.\mathbf{Z} \mid x > 0$) muss (MUST) erhöht werden,
 - > wenn ausschließlich API-kompatible Bugfixes eingeführt werden.
- + Die Minor Version Y ($x.\mathbf{Y}.z \mid x > 0$) muss (MUST) erhöht werden,
 - > wenn neue Funktionalitäten, welche kompatibel zur bisherigen API sind, veröffentlicht werden.
 - > wenn eine Funktion der öffentlichen API als deprecated markiert wird.
 - > Die Patch Version muss (MUST) dann auf Null zurückgesetzt werden.
- + Die Major Version X ($\mathbf{X}.y.z \mid X > 0$) muss (MUST) immer dann erhöht werden,
 - > wenn API-inkompatible Änderungen in die öffentlichen API eingeführt werden.
 - > Wenn diese Versionsnummer erhöht wird, muss (MUST) sowohl die Minor Version als auch die Patch Version auf Null zurückgesetzt werden.
- +

"<https://semver.org/lang/de/>"

Node Module folgen der Semantischen Versionierung

Express

Express Basis Applikation

```
import express from 'express';
var express = require('express');

const app = express();

app.get('/', function(request, response){
    response.send('hello world');
});

app.listen(3000);
```

Die ersten App-Methoden

app.get, app.post, app.put, ...,
app.all,
app.param
app.route
app.render
app.engine

- Routing HTTP requests.
- Configuring middleware.
- Rendering HTML views.
- Registering a template engine.

Routes

```
app.get('/', callback);
```

Die Pfadangabe ('/') kann

- ein Stringausdruck,
- ein Pfadmuster,
- ein regulärer Ausdruck,
- oder ein Array mit einer Kombination

daraus sein.

Das Request Objekt

- + bildet den HTTP Request ab.
- + Es enthält Eigenschaften wie den Querystring, übergebene Parameter, den Request-Body, die HTTP-Headers und so weiter.
- + <http://expressjs.com/de/4x/api.html#req>

Das Request Objekt

```
app.get('/user/:id', function(request, response) {  
  res.send('user ' + request.params.id);  
});  
  
console.log('The views directory is ' +  
request.app.get('views'));  
  
console.log(request.baseUrl);  
  
console.log(request.cookies.name);  
  
console.log(request.hostname);  
console.log(request.ip);  
console.log(request.method);  
console.log(request.path);  
console.log(request.query);  
console.log(request.route);
```

Request Body

```
var app          = require('express')();
var bodyParser = require('body-parser');
var multer     = require('multer'); // v1.0.5
var upload      = multer(); // for parsing
                           multipart/form-data

// for parsing application/json
app.use(bodyParser.json());

// for parsing application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

app.post('/profile', upload.array(), function (request,
response, next) {
  console.log(request.body);
  response.json(request.body);
});
```

Das Response Objekt

- + Bildet die HTTP Antwort, die Express bei einer Anfrage formuliert und abschickt.
- + Es besitzt ausser einigen Eigenschaften vor allem die Methode, um eine Antwort zu bilden.

Das Response Objekt

```
response.append('Link', ['<http://localhost/>', '<http://  
localhost:3000/>']);  
response.append('Set-Cookie', 'foo=bar; Path=/; HttpOnly');  
response.append('Warning', '199 Miscellaneous warning');  
  
response.attachment('path/to/logo.png');  
  
response.cookie('name', 'tobi', { domain: '.example.com', path: '/  
admin', secure: true });  
  
response.download('/report-12345.pdf');  
  
res.send(new Buffer('whoop'));  
res.send({ some: 'json' });  
res.send('<p>some html</p>');  
res.status(404).send('Sorry, we cannot find that!');  
res.status(500).send({ error: 'something blew up' });  
  
response.end();  
response.status(404).end();
```

```
response.json(null);
response.json({ user: 'tobi' });
response.status(500).json({ error: 'message' });

response.jsonp(null);
response.jsonp({ user: 'tobi' });
response.status(500).jsonp({ error: 'message' });
```

Response Formate

```
res.format({
  'text/plain': function(){
    res.send('hey');
  },
  'text/html': function(){
    res.send('<p>hey</p>');
  },
  'application/json': function(){
    res.send({ message: 'hey' });
  },
  'default': function() {
    // log the request and respond with 406
    res.status(406).send('Not Acceptable');
  }
});
```

Live Reload

Live Reload im Browser

```
npm install connect-livereload --save-dev
npm install livereload --save-dev

// -----
import connectLiveReload from 'connect-livereload';
import liveReload from 'livereload';

const liveReloadServer = liveReload.createServer();
liveReloadServer.server.once('connection', () => {
  setTimeout(() => {
    liveReloadServer.refresh("/")
  }), 100
});

...
// use with express
app.use(connectLiveReload({
  port: 35729
}));
```

Utilities Klasse

```
util.debugLog(section)
util.deprecate(function, string)
util.format(format[, ...args])
util.inherits(constructor, superConstructor)
util.inspect(object[, options])
```

```
%s - String.  
%d - Number (both integer and float).  
%j - JSON. Replaced with the string '[Circular]' if  
the argument contains circular references.  
%% - single percent sign ('%'). This does not  
consume an argument.  
  
util.format('%s:%s', 'foo', 'bar', 'baz'); //  
'foo:bar baz'
```

```
dataString = util.format(  
    '%s-%s-%s %s:%s:%s Lorem ipsum dolor sit.\n',  
    now.getFullYear(),  
    (now.getMonth() + 1),  
    now.getDate(),  
    now.getHours(),  
    now.getMinutes(),  
    now.getSeconds()  
);
```

Ereignisse verarbeiten

Eventhandler und -listener

- + Viele Objekte senden in node Events aus: ein **net.server** sendet einen Event, wenn sich ein Client verbindet. Ein **fs.readStream** sendet ein Event, wenn die Datei geöffnet ist.
- + Jedes Objekt, das Events sendet ist eine Instance des **events.EventEmitter Konstruktors**. Diese kann über **require("events");** eingebunden werden.
- + Funktionen können an Objekte gebunden werden, wenn diese einen Event gesendet haben: Diese Funktionen werden Eventhandler genannt.
- + In Eventhandlern verweist **this** auf das Event - aussendende Objekt.

Eventlistener setzen

```
emitter.addListener(event, listener)  
emitter.on(event, listener)  
  
server.on('connection', function (stream) {  
    console.log('someone connected!');  
});
```

Einmal - Event

```
emitter.once(event, listener)

server.once('connection', function (stream) {
  console.log('Oh, a first user!');
});
```

Löschen von Eventlistenern

```
emitter.removeListener(event, listener)

var callback = function(stream) {
  console.log('someone connected!');
};

server.on('connection', callback);
// ...
server.removeListener('connection', callback);
```

Weiteres ...

```
emitter.removeAllListeners([event])
```

```
emitter.setMaxListeners(n) // 0 == unbegrenzt
```

Dateisystem

readFile

```
fs.readFile(filename, function(error, content) {  
  if (error) {  
    throw error;  
  };  
  console.log(content.toString());  
};);
```

writeFile

```
function() {
  // a - append, ggf. new
  fs.open(logFile, 'a', function(error, handle) {
    if (error) {
      throw error;
    };

    let
      logFileString = null,
      buffer = null;

    logFileString = (i++) + ': Ipsum Lorem Nullam Egestas\n';

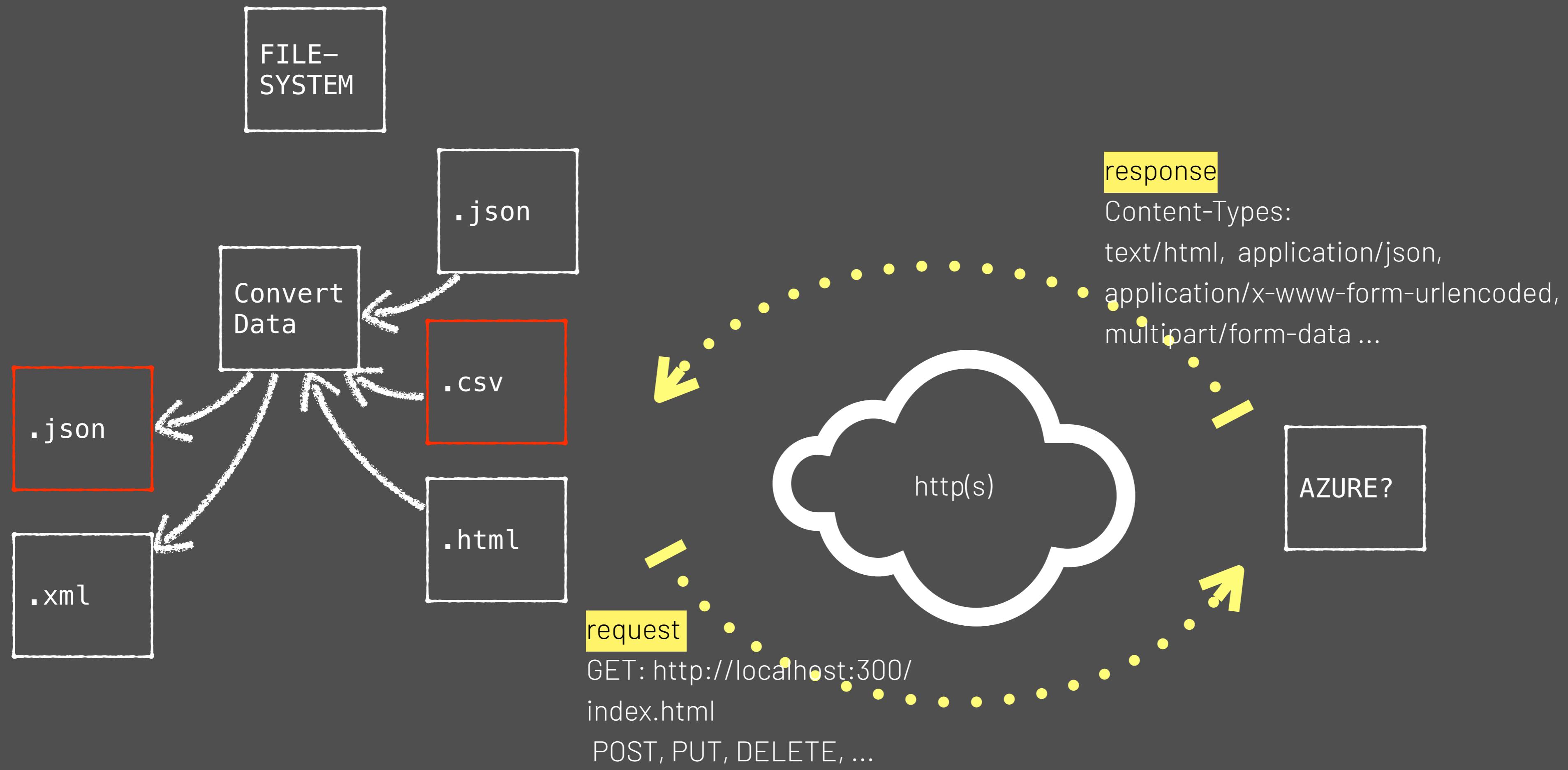
    buffer = new Buffer(logFileString);

    fs.write(handle, buffer, 0, buffer.length, null, function() {
      fs.close(handle, function() { /* idle */ });
    });

  });
};
```

Watchfile

```
fs.watchFile(logFile, function(currentTime,  
previousTime) {  
    // atime - access time  
    // mtime - modify time  
    // ctime - current time (?)  
    console.log('current time is ' +  
currentTime.mtime);  
    console.log('previous time is ' +  
previousTime.mtime);  
  
});
```



Anwendungsfall Filesystem

Buffer

Buffer (content to Filesystem to content)

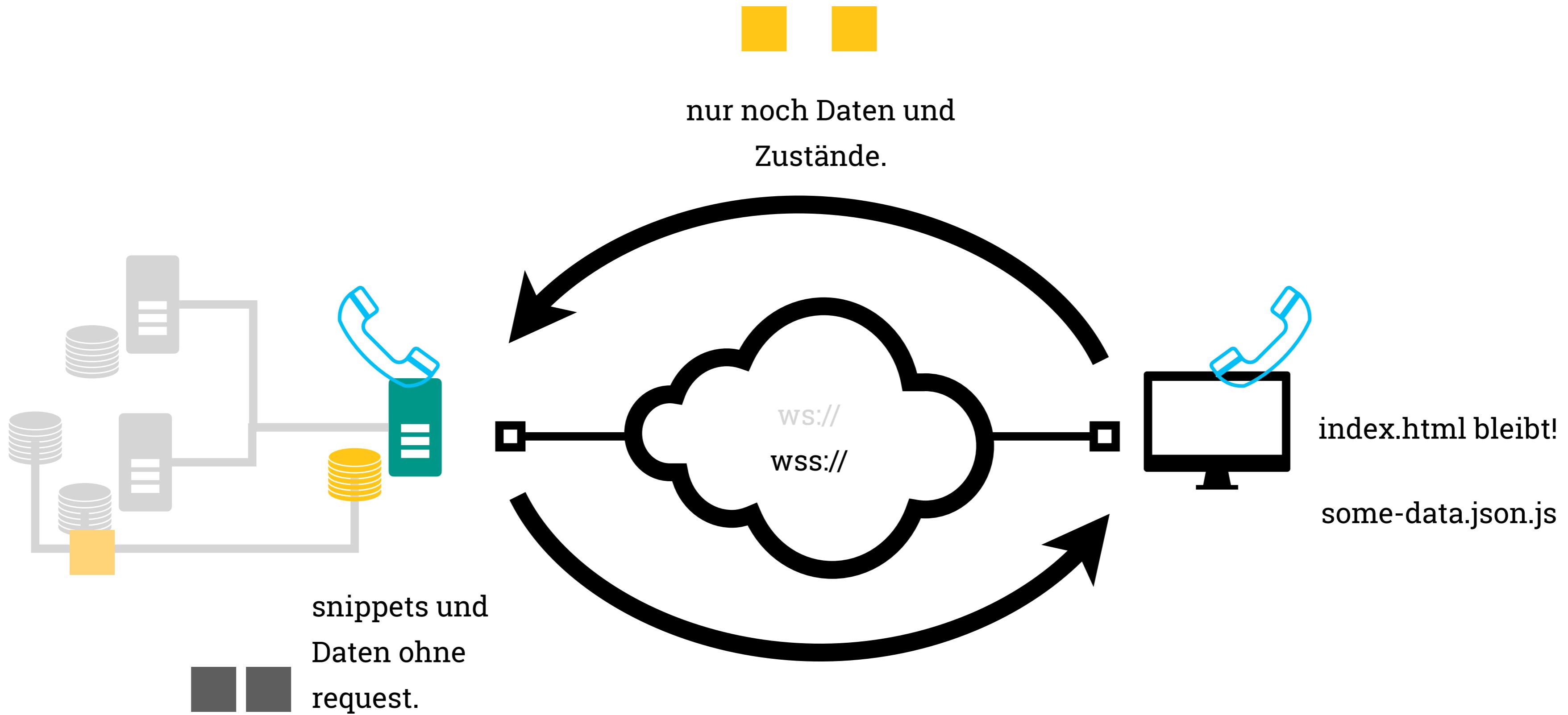
```
// read a file
onReadFile = function (error, buffer) {
    if (error) {
        throw error;
    }
    console.log(buffer.toString());
}

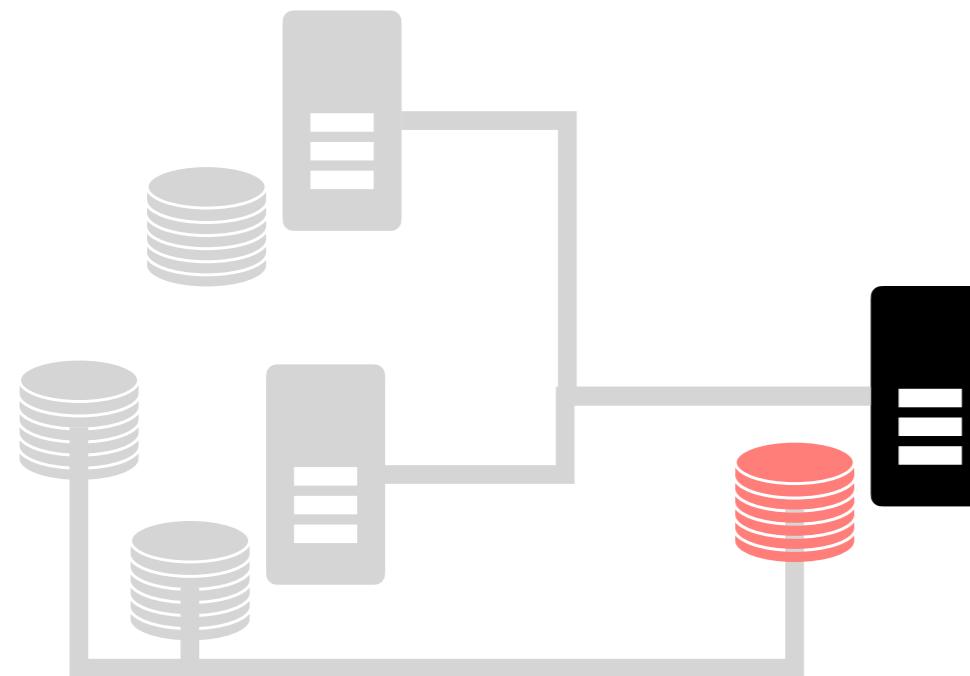
// some content
string = getDateString() + ' Lorem ipsum ...\\n';

// string to buffer
buffer = Buffer.from(string);
```

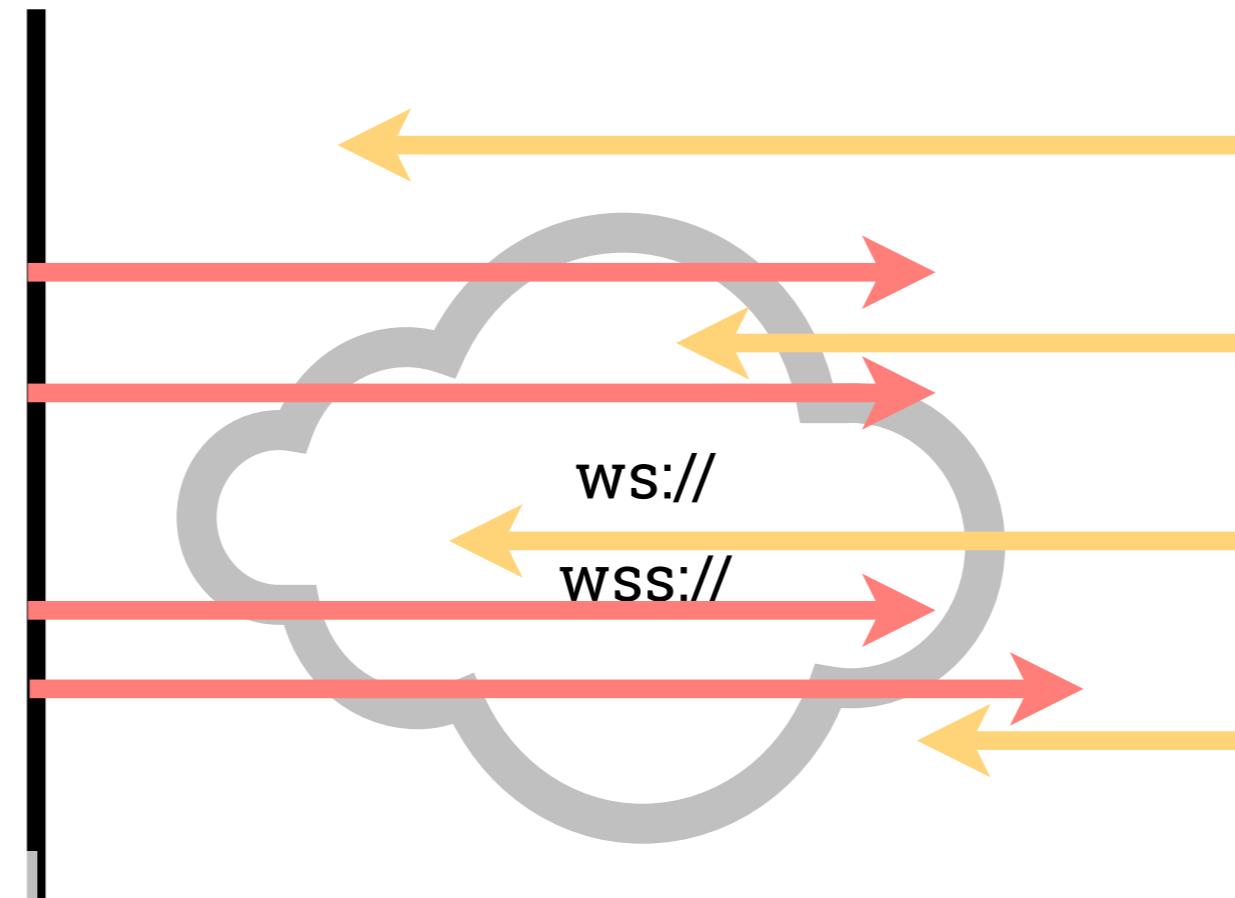
Websockets

Einführung in Websockets





snippets und
Daten ohne
request.



nur noch Daten und
Zustände.



index.html bleibt!
login.php
new-image.png
some-data.json.js

„Mit AJAX Requests kann ich über den
Browser keine Modelleisenbahn steuern.“

-Jan Hickson

Websockets

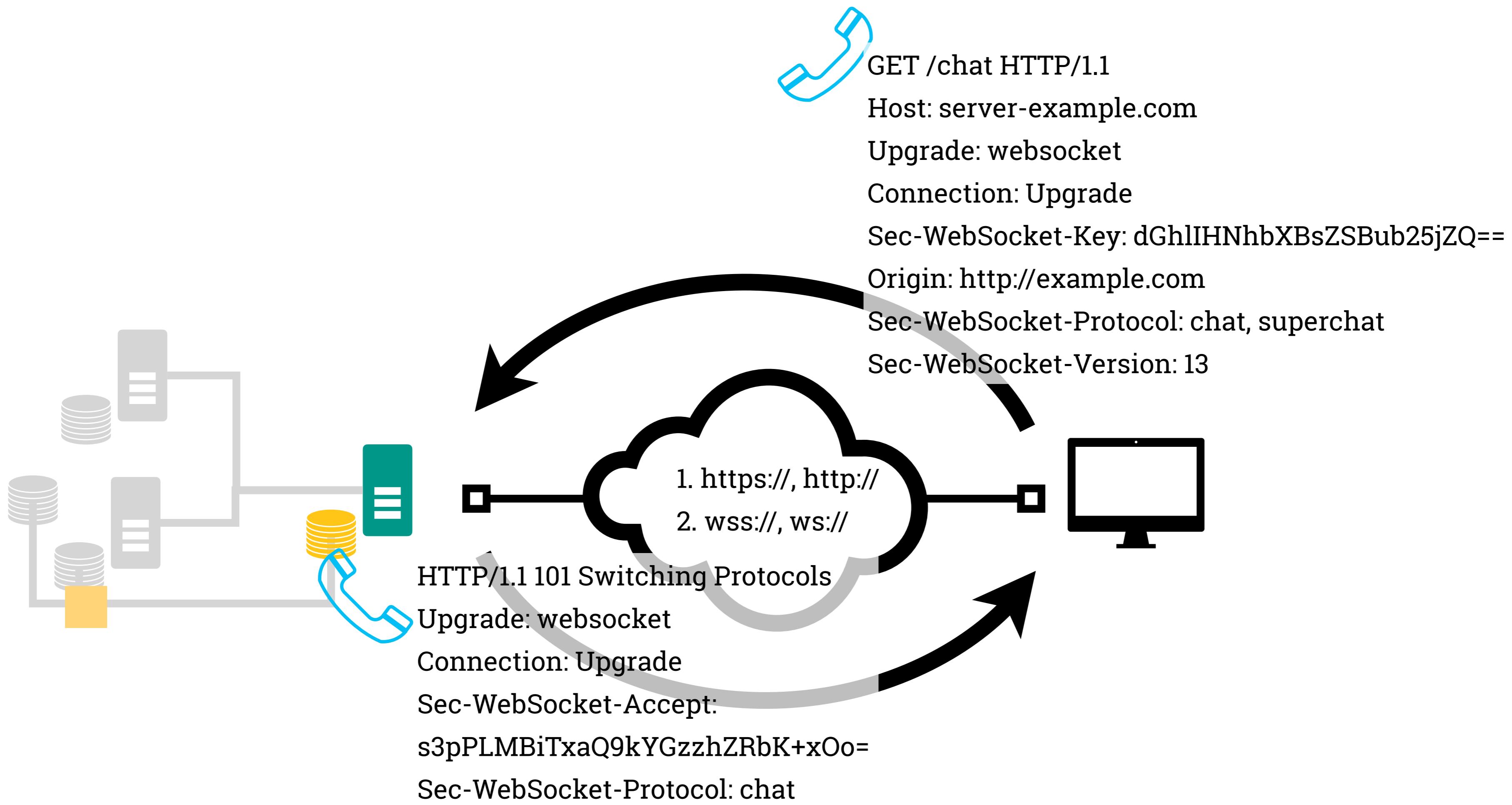
- + ein Protokoll, das eine persistente Verbindung zwischen Browser und Webserver offenhält.
- + ws:// - WebSocket Protokoll
wss:// - WebSocket Secure Protokoll

Bidirektional & Full duplex

- + Über diese Verbindung kann in beiden Richtungen (Client <--> Server) kommuniziert werden.
- + Aufgrund des binären Protokolls (ws:/wss:) hat es sehr wenig Overhead.

Http Initialisierung

- + Der initiale Verbindungsaufbau einer Websocket-Verbindung läuft über HTTP (oder HTTPS),
- + Ein Upgrade-Header teilt dem Server mit, dass auf das Websocket-Protokoll „upgegradet“ werden soll.



Quelle: The WebSocket Protocol; Fette & Melnikov

Server Proxy in nodejs

- + Serverseitig wird ein Socketproxy bereitgestellt, der z. B. mit wenigen Zeilen in node.js aufgesetzt werden kann.
- + nodejs beherrscht dazu HTTP und kann einen Upgrade-Headers verarbeiten.

Head eines HTTP-Request

```
GET /PollingStock//PollingStock HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5) Gecko/
20091102
Firefox/3.5.5

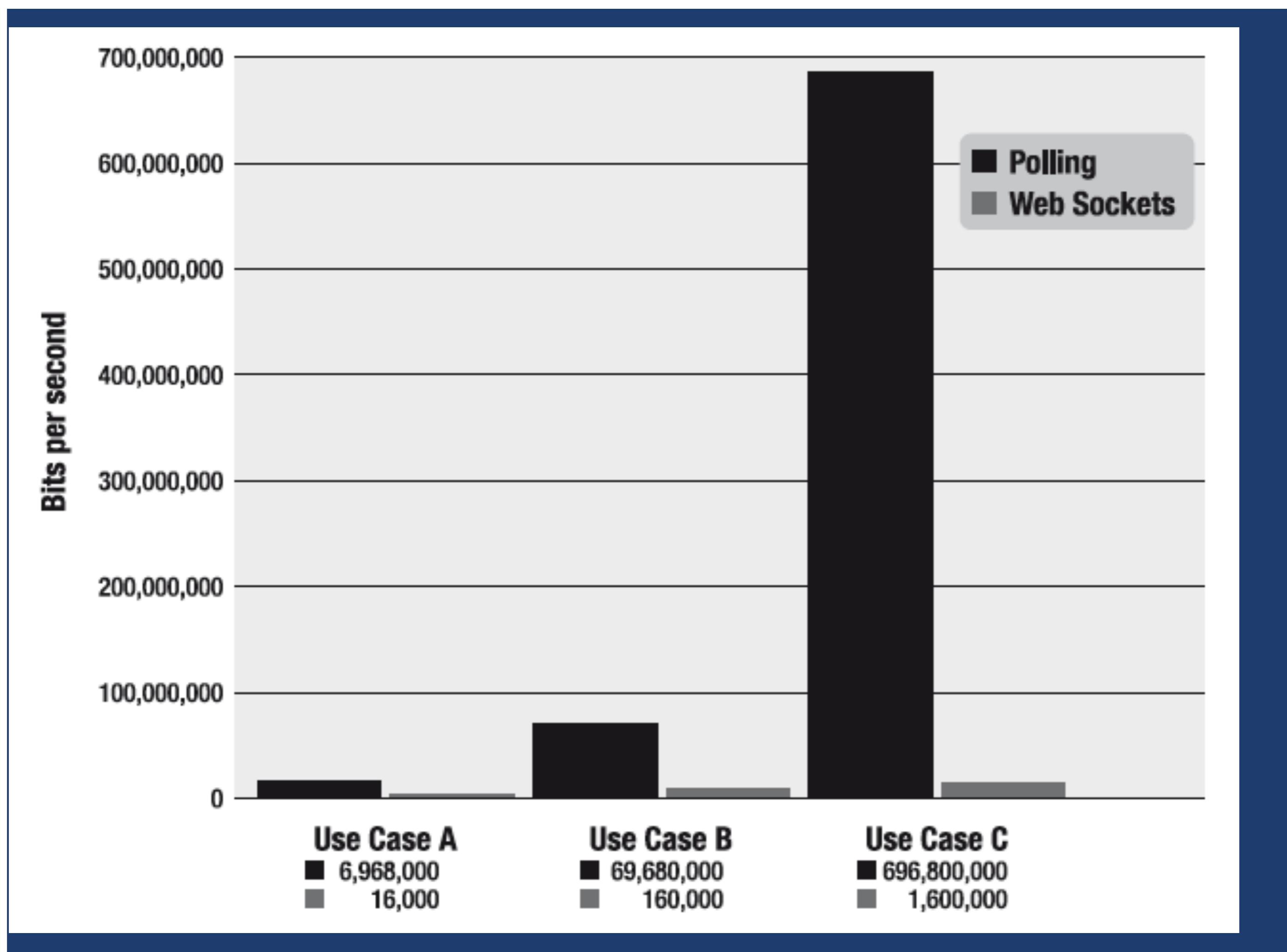
Upgrade: WebSocket?

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/PollingStock/
Cookie: showInheritedConstant=false;
showInheritedProtectedConstant=false;
showInheritedProperty=false; showInheritedProtectedProperty=false;
showInheritedMethod=false; showInheritedProtectedMethod=false;
showInheritedEvent=false; showInheritedStyle=false; showInheritedEffect=false
```

Head eines Socket-Request



\0x00 Hello, WebSocket \0xff



Nodejs Sockets

Ein Socketproxy

```
// Websocket-Server (-> npm install ws!)

var WebSocketServer = require('ws').Server
var wss = new WebSocketServer({
    host: „192.168.2.1“,
    port: 8000
});

wss.on('connection', function(ws) {
    console.log('client verbunden...');

    ws.on('message', function(message)  {
        console.log('von Client empfangen: ' + message);
        ws.send('von Server empfangen: ' + message);
    });
});
```

Der Browserpart zur Socketkommunikation

```
function connect() {
    // Websocket
    var socket = new WebSocket("ws://192.168.2.1:8000");

    socket.onopen = function() {
        console.log("Socket Status: "
            + socket.readyState + " (open)");
    }

    socket.onmessage = function(msg) {
        console.log("Empfangen: " + msg.data);
    }

    socket.onerror = function (err) {
        console.log("Ein Fehler ist aufgetreten.");
    }

    socket.send("Hallo Welt");
}
```

Die Javascript API für den Client

Das readyState Attribut enthält den Status der Verbindung

CONNECTING (numeric value 0)

Die Verbindung wurde noch nicht hergestellt.

OPEN (numeric value 1)

Die Verbindung steht, Kommunikation ist möglich.

CLOSING (numeric value 2)

Die Verbindung führt den Closing Handshake aus.

CLOSED (numeric value 3)

Die Verbindung wurde geschlossen oder konnte nicht hergestellt werden.

Die Websocket Methoden

```
mySocket = new WebSocket();

mySocket.onopen = function(evt) {
    console.log(„Connection open ...“);
};

mySocket.onmessage = function(evt) {
    console.log( "Received Message: " + evt.data);
};

mySocket.onclose = function(evt) {
    console.log("Connection closed.");
};

mySocket.onerror = function(evt) {
    console.log("An error happened.");
};
```

Socket öffnen und Daten senden

```
var mySocket = new WebSocket('ws://  
game.example.com:12010/updates');  
  
mySocket.onopen = function () {  
  
    setInterval(function() {  
  
        if (mySocket.bufferedAmount === 0) {  
            mySocket.send( );  
        }  
  
    }, 50);  
  
};
```

onmessage

```
mySocket.onmessage = function (event) {  
  
    if (event.data === 'on') {  
        turnLampOn();  
    } else if (event.data === 'off') {  
        turnLampOff();  
    }  
  
};
```

socket.io

Die socket.io package.json

```
{  
  "name": "socket-chat-example",  
  "version": "0.0.1",  
  "description": "my first socket.io app",  
  "dependencies": {  
    "express": "^4.18.1",  
    "socket.io": "^4.5.1"  
  }  
}
```

Chat-Beispiel - import und Webserver

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const {
  Server
} = require("socket.io");

const io = new Server(server);

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
```

Chat-Beispiel: Message-Handling

```
io.on('connection', (socket) => {
    console.log('a user connected');

    socket.on('disconnect', () => {
        console.log('user disconnected');
    });

    socket.on('chat message', (msg) => {
        io.emit('chat message', msg);
    });
};

server.listen(3000, () => {
    console.log('listening on *:3000');
});
```

Chat: HTML-Scaffolding

```
<!DOCTYPE html>
<html>

<head>
    <title>Socket.IO chat</title>
    <style> ... </style>
</head>

<body>
    <ul id="messages"></ul>
    <form id="form" action="">
        <input id="input" autocomplete="off" /><button>Send</button>
    </form>

    <script src="/socket.io/socket.io.js"></script>
    <script>
        var socket = io();
        ...
    </script>
</body>

</html>
```

Chat-Stylesheet

```
body {  
    margin: 0;  
    padding-bottom: 3rem;  
    font-family: sans-serif;  
}  
  
#form {  
    background: rgba(0, 0, 0, 0.15);  
    padding: 0.25rem;  
    position: fixed;  
    bottom: 0;  
    left: 0;  
    right: 0;  
    display: flex;  
    height: 3rem;  
    box-sizing: border-box;  
    backdrop-filter: blur(10px);  
}  
  
#input {  
    border: none;  
    padding: 0 1rem;  
    flex-grow: 1;  
    border-radius: 2rem;  
    margin: 0.25rem;  
}  
  
#input:focus {  
    outline: none;  
}  
#form>button {  
    background: #333;  
    border: none;  
    padding: 0 1rem;  
    margin: 0.25rem;  
    border-radius: 3px;  
    outline: none;  
    color: #fff;  
}  
  
#messages {  
    list-style-type: none;  
    margin: 0;  
    padding: 0;  
}  
  
#messages>li {  
    padding: 0.5rem 1rem;  
}  
#messages>li:nth-child(odd) {  
    background: #efefef;  
}
```

Chat-Script: Form und Message-Handling

```
var form = document.getElementById('form');
var input = document.getElementById('input');

form.addEventListener('submit', function (e) {
    e.preventDefault();
    if (input.value) {
        socket.emit('chat message', input.value);
        input.value = '';
    }
});

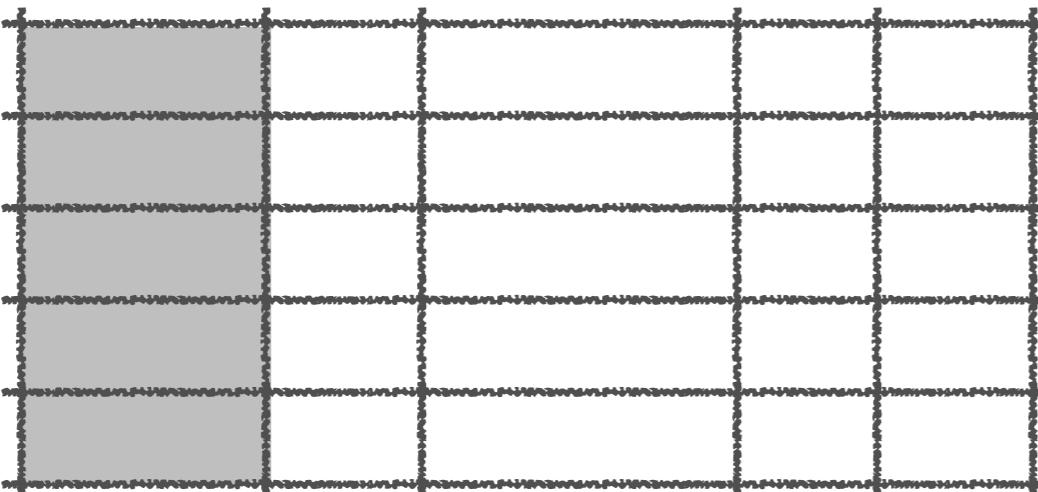
socket.on('chat message', function (msg) {
    var item = document.createElement('li');
    item.textContent = msg;
    messages.appendChild(item);
    window.scrollTo(0, document.body.scrollHeight);
});
```

Mongodb

SQL vs. Not only SQL - noSQL

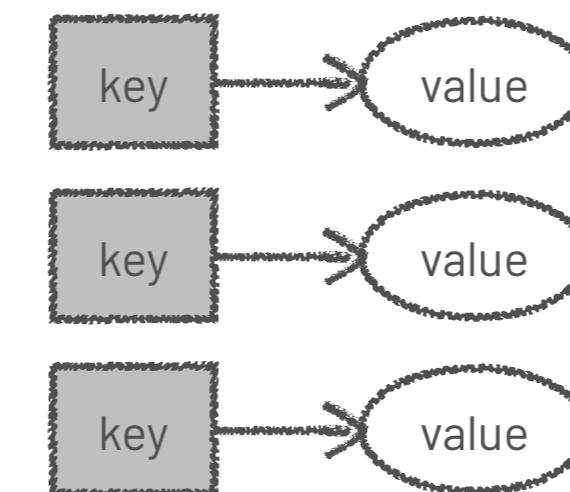
SQL

Relational

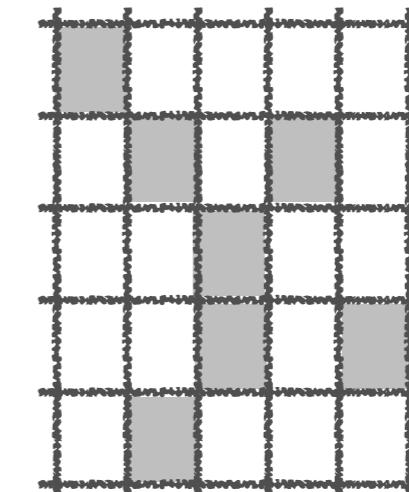


noSQL

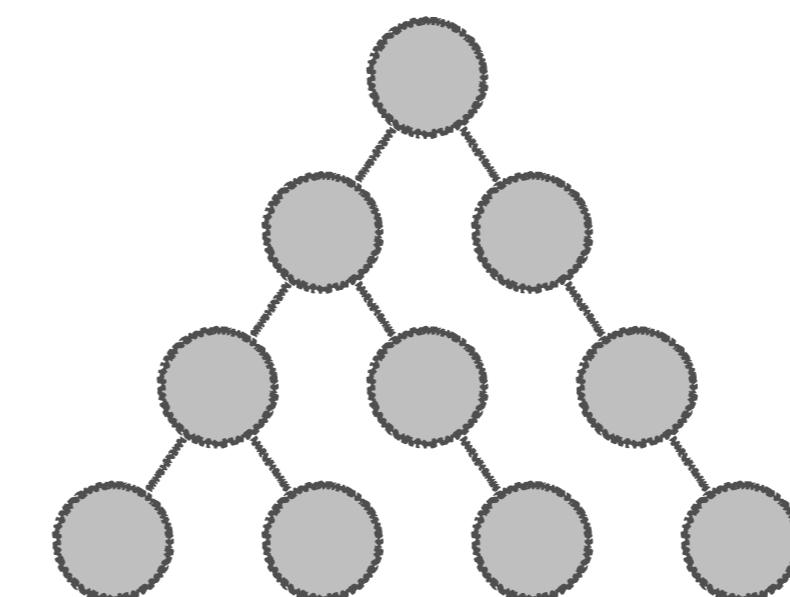
key-value
(i.e. localStorage)



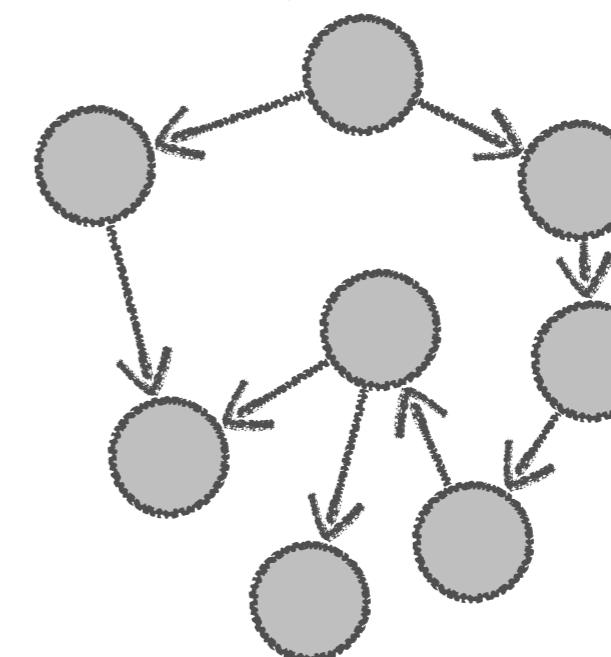
wide-column,
i.e. Azure tables



document store,
i. e. Mongo

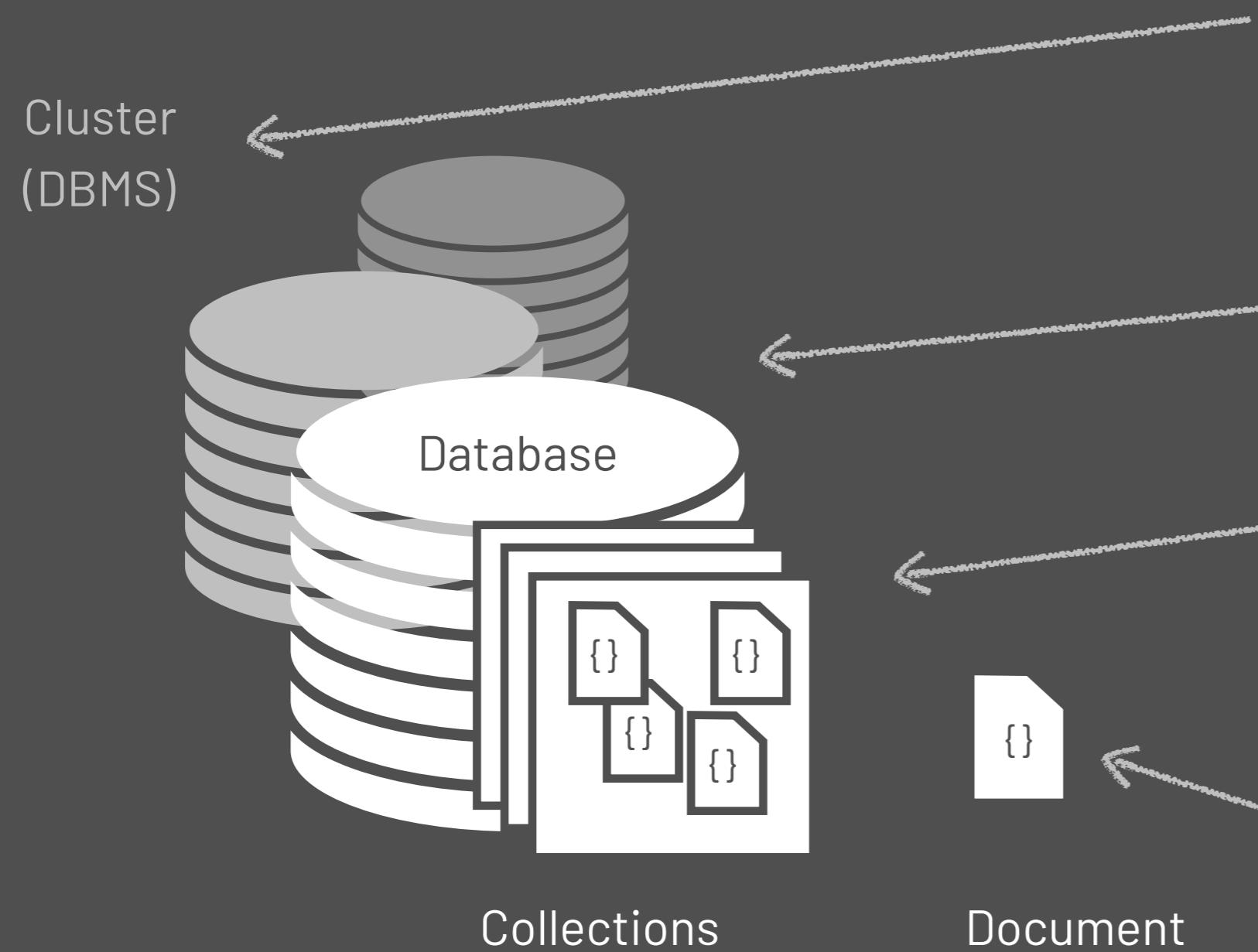


graph,
i.e. Neo4j



Dokumentenorientierte Datenbanken

- + speichern Informationen in einfachen Tabellenstrukturen und Dokumenten.
- + heissen auch *document stores*.
- + *document stores* sind nicht relational.
- + Die Daten sind teilstrukturiert. Sie folgen keiner festen Struktur, sondern tragen die Struktur in sich selbst.
- + Geordnet wird an Hand von Markierungen.



```
client = new MongoClient(uri);  
client.connect();
```

```
database = client.db("my_db");
```

```
collection = database.collection("autor");  
collection.insertOne({ ... });
```

```
document = collection.findOne(query, options);  
documents = collection.findMany( ... );
```

```
document = collection.find( ... )  
.sort( ... )  
.limit( ... );
```

Mongo-DB Begriffe

Collections in document stores

Collection: »Einträge«

```
[{"id": 1,  
 "autor": "Michael",  
 "datum": "2015-03-23T11:19:21.000Z",  
 "ueberschrift": "Mein neuer Blog",  
 "eintrag": "Hier der Text des Blogs...",  
 "markierung": ["Allgemein", "News"],  
 "kommentare": [  
     {"autor": "klaus1",  
      "datum": "2015-03-24T11:23:15.000Z",  
      "kommentar": "Super Blog!"},  
     {"autor": "tom15",  
      "datum": "2015-03-25T11:29:15.000Z",  
      "kommentar": "Unbedingt lesen!"}],  
  
    {"id": 2,  
     "autor": "Oliver",  
     ...}  
}
```

Collection: »Benutzer«

```
{"benutzer": "Michael",  
 "name": {  
     "vorname": "Michael",  
     "nachname": "Schwarze"},  
 "password": "$5$6&7665!!223/34%4",  
  
 "benutzer": "tom15",  
 "password": ...  
 ...}
```

Die Daten sind abfrage-orientiert gespeichert, ggf. redundant und lassen keine Beziehungen (JOIN) zu.

nodejs package installieren

```
npm install mongodb
```

Einbinden und Zugangsdaten zum Cluster

```
const {
  MongoClient
} = require('mongodb');

const uri = "mongodb+srv://
  username:A2fPQYctCXDX7PQd
  @mongocluster.123.mongodb.net/
  test?retryWrites=true&w=majority";
```

Liste der Datenbanken abfragen

```
async function main() {  
  const client = new MongoClient(uri);  
  
  async function listDatabases(client) {  
    listDatabases = await client.db().admin().listDatabases();  
  
    console.log("Databases:");  
    listDatabases.databases.forEach(db => console.log(` - ${db.name}`));  
  };  
  
  try {  
    await client.connect(); // Connect to the MongoDB cluster  
  
    await listDatabases(client); // Make the appropriate DB calls  
  
  } catch (e) {  
    console.error(e);  
  } finally {  
    await client.close();  
  }  
}  
  
main().catch(console.error);
```

Eine Collection erstellen

```
const client = new MongoClient(uri);
try {
  await client.connect();

  const database  = client.db("sample_mflix");
  const collection = database.collection("movies");

  // Query for a movie that has the title 'The Room'
  const query = {
    title: "The Room"
  };

  const options = {
    // sort matched documents in descending order by rating
    sort: {
      "imdb.rating": -1
    },
    // Include only the `title` and `imdb` fields in the returned document
    projection: {
      _id: 0,
      title: 1,
      imdb: 1
    }
  };
}
```

In der Collection nach Eigenschaft finden

```
// Find a Document  
coll.findOne({ title: 'Hamlet' });
```

```
// Find Multiple Documents  
coll.find({ year: 2005 });
```

Ein Dokument in die Collection einfügen

```
// Insert a Document  
coll.insert({ title: 'Jackie Robinson' });
```

MySQL

A Database Connection

```
npm install mysql;

'use strict';
// -----
const mysql = require('mysql');
const dbConfig = require('./db-config');

let
  db = mysql.createConnection(dbConfig),
  sql = null;

db.on('error', function(error) {
  console.log(error);
});

...
db.end();
```

The Database Connection Data

```
// db-config.json
{
    "host":      "localhost",
    "user":      "root",
    "password":  "root",
    "database":  "application",
    "port":      3206,

    "more": {
        "table": "user"
    }
}
```

SQL Queries

```
db.query('CREATE DATABASE IF NOT EXISTS application;');
db.query('USE application;');

// Tabelle anlegen
db.query('DROP TABLE IF EXISTS user;');

sql = "CREATE TABLE user ( " +
    "userId INT(11) AUTO_INCREMENT, " +
    "username VARCHAR(50), " +
    "email VARCHAR(50), " +
    "password VARCHAR(50), " +
    "PRIMARY KEY (userId) );";
db.query(sql);

sql = "INSERT INTO user " +
    "(username, email, password) " +
    "VALUES " +
    "('Michael', 'michael@zenbox.de', 'geheim')," +
    "('Paula', 'paula@zenbox.de', 'geheim')," +
    "('Klaus', 'klaus@zenbox.de', 'geheim');";

db.query(sql, function() {
    console.log('Datensätze geschrieben.');
});
```

MySQL in an index route 1

Connecting the database

```
'use strict';
// -----
const express = require('express');
const router = express.Router();
const mysql = require('mysql');

let
  dbConfig = require('../db-config.json'),
  db = null,
  query = null;

// connect the database service
dbConfig.database = dbConfig.more.database;
db = mysql.createConnection(dbConfig);

...
module.exports = router;
```

MySQL in an index route 2

A get request Query

```
router.get('/', function (request, response) {  
  query = "SELECT * FROM user WHERE 1;";  
  db.query(query, function (error, result) {  
    if (error) { console.dir(error); process.exit(0); }  
    // console.dir(result);  
    response.render(  
      'index',  
      {  
        title : ' a mysql result',  
        result : result  
      }  
    );  
  });  
});
```

Output Result as Table

```
<table>
  <thead>
    <tr>
      <% for (let key in result[0]) { %>
        <th>
          <%= key %>
        </th>
      <% } %>
    </tr>
  </thead>
  <tbody>
    <% for (let i=0, len=result.length; i<len; i+=1) { %>
      <tr>
        <% for (let key in result[i]) { %>
          <td>
            <%= result[i][key] %>
          </td>
        <% } %>
      </tr>
    <% } %>
  </tbody>
</table>
```

Crypto

Clpher / Decipher

Symmetric Encryption

```
let
  crypto = require('crypto'),
  algorithm = 'aes-192-cbc',
  password = 'Password used to generate key',
  salt = 'salt',
  key, iv, cipher, decipher, encrypted, decrypted;

key = crypto.scryptSync(password, salt, 24);
iv = Buffer.alloc(16, 0);

// Cipher
cipher = crypto.createCipheriv(algorithm, key, iv);

cipher.on('readable', () => {
  let chunk;
  while (null !== (chunk = cipher.read())) {
    encrypted += chunk.toString('hex');
  }
});
cipher.on('end', () => {
  console.log(encrypted);
});

cipher.write('some clear text data');
cipher.write('More vulputate Purus');
cipher.end();
```

Cipher / Decipher

Symmetric Encryption

```
// Decipher
decipher = crypto.createDecipheriv(algorithm, key, iv);

decipher.on('readable', () => {
  while (null !== (chunk = decipher.read())) {
    decrypted += chunk.toString('utf8');
  }
});

decipher.on('end', () => {
  console.log(decrypted);
});

decipher.write(encrypted, 'hex');
decipher.end();
```

Clpher / Decipher

Symmetric Encryption

```
// Using the .update() and .final() methods:  
  
// Cipher  
cipher = crypto.createCipheriv(algorithm, key, iv);  
encrypted = cipher.update('some clear text data',  
'utf8', 'hex');  
encrypted += cipher.final('hex');  
console.log(encrypted);  
  
// Decipher  
decipher = crypto.createDecipheriv(algorithm, key, iv);  
decrypted = decipher.update(encrypted, 'hex', 'utf8');  
decrypted += decipher.final('utf8');  
console.log(decrypted);
```

ReST API

Was ist ReST?

- + ReST ist ein Akronym für *Representational State Transfer*.
- + Es basiert auf Web-Standards und dem HTTP-Protokoll
- + Die REST Architektur beschreibt sechs Vorschriften (nach Roy Fielding)
 1. Uniform Interface (Einheitliche Schnittstelle)
 2. Stateless (Zustandslosigkeit)
 3. Cacheable (speicherbar, wiederverwendbar)
 4. Client-Server-Modell
 5. Layered System (mehrschichtig, hierarchisch)
 6. Code on Demand (optional)

Client-Server-Modell

- + REST verlangt ein Client-Server-Modell, will also das Nutzerinterface von der Datenhaltung getrennt sehen.
- + Damit lassen sich Clients einfach auf verschiedene Plattformen portieren.
- + (U. a. so) vereinfachte ServerKomponenten lassen sich gut skalieren.

Zustandslosigkeit

- + Client und Server müssen zustandslos („stateless“) miteinander kommunizieren.
- + Jede Anfrage eines Clients beinhaltet alle Informationen, die ein Server benötigt.
- + Server können auf keinen gespeicherten Kontext zurückgreifen.
- + Diese Einschränkung verbessert Visibility, Zuverlässigkeit und Skalierbarkeit.
- + Dafür nimmt REST Nachteile bei der Netzwerkperformanz in Kauf.
- + Und Server haben keine Kontrolle über ein konsistentes Verhalten der Client-App.

Caching

- + Um die Netzwerkeffizienz zu verbessern, können Clients vom Server gesendete Antworten auch speichern und bei gleichartigen Requests später erneut verwenden.
- + Die Vorteile responsiverer Anwendungen mit höherer Effizienz und Skalierbarkeit werden so mit dem Risiko erkauft, dass Clients auf veraltete Daten aus dem Cache zurückgreifen.
- + Die Informationen müssen dem entsprechend als „cacheable“ oder „non-cacheable“ gekennzeichnet werden.

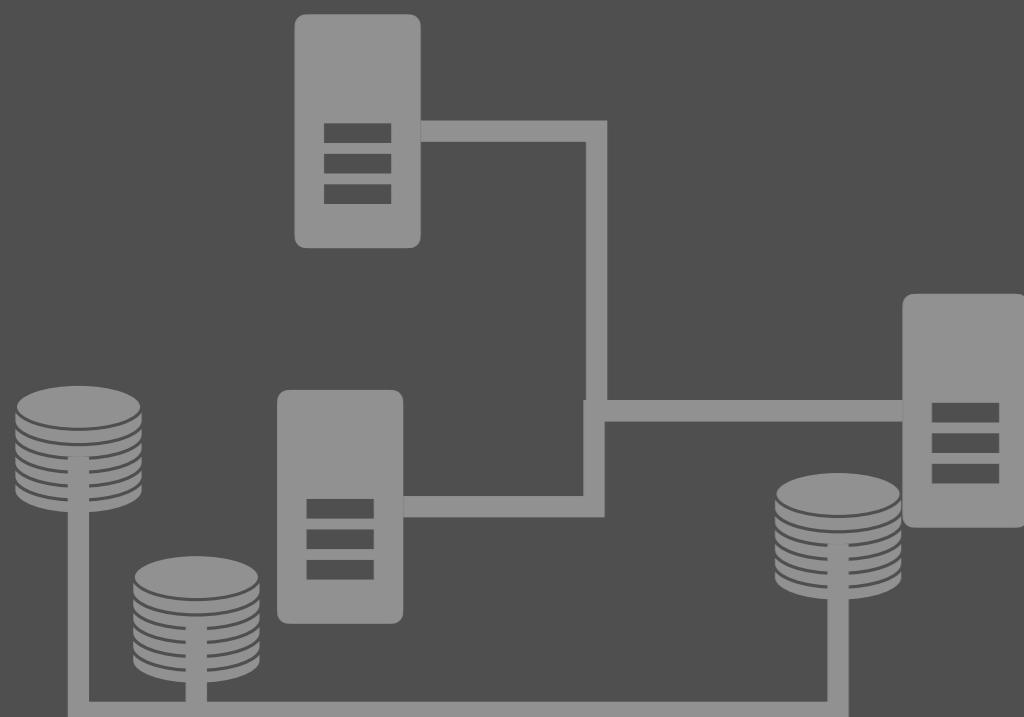
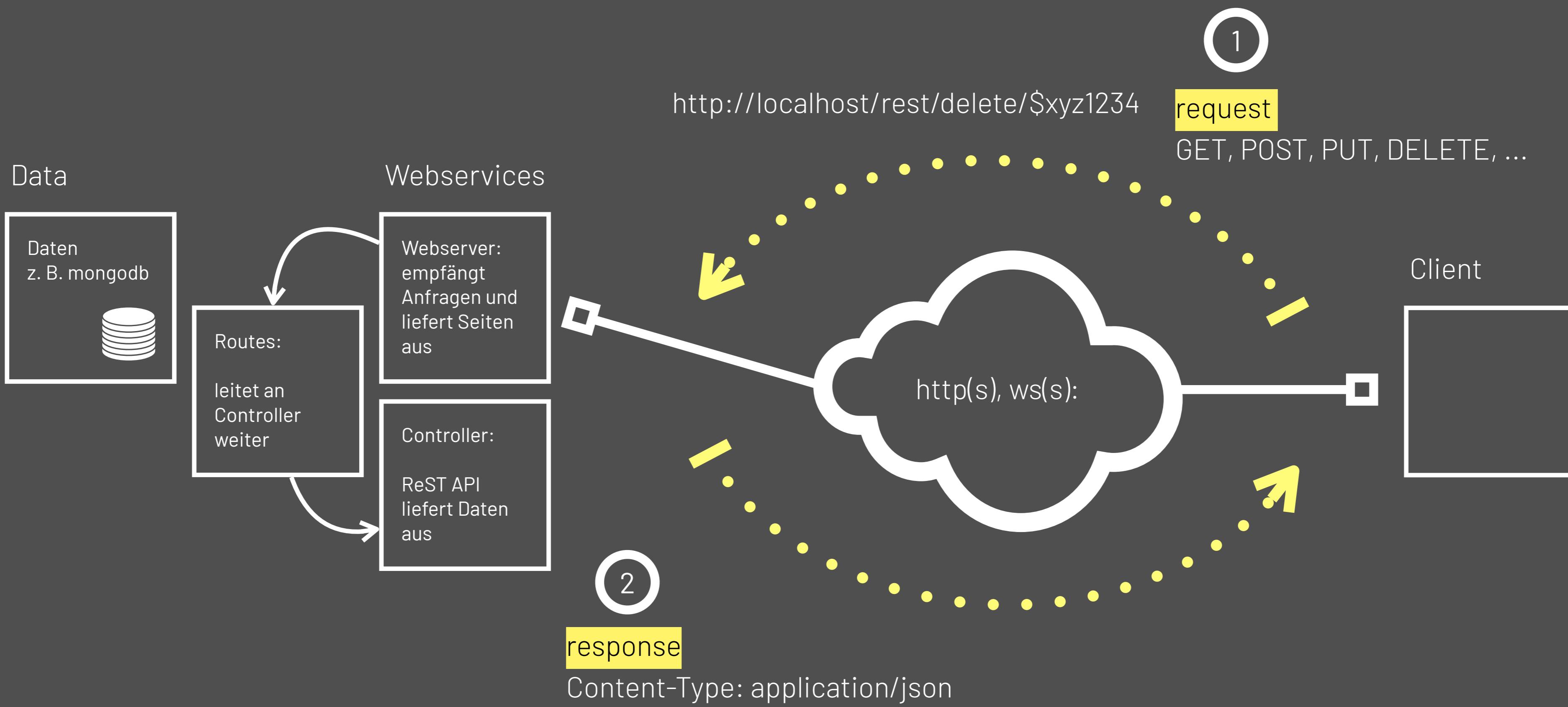
Layered System

- + REST setzt auf mehrschichtige, hierarchische Systeme („layered system“).
- + Jede Komponente kann ausschließlich jeweils direkt angrenzende Schichten sehen. Somit lassen sich beispielsweise Legacy-Anwendungen kapseln.
- + Als Load Balancer agierende Vermittler („Intermediaries“) können die Skalierbarkeit verbessern.
- + Als Nachteile dieses Constraints gelten ein zusätzlicher Overhead und erhöhte Latenzen.

Code-On-Demand

- + Dieses Constraint fordert, dass die Funktionen von Clients über nachlad- und ausführbare Programmteile erweitert werden können – etwa in Form von Applets oder Skripten.
- + Als optionales Constraint kann diese Bedingung in bestimmten Kontexten jedoch deaktiviert sein.

- + ReST-Anwendungen verwenden HTTP Requests, um die vier CRUD Operationen auszuführen.
- + C: create, R: read, U: update, und D: delete).
- + GET - fordert Daten vom Server an
- + POST - übermittelt Daten an den Server
- + PUT/PATCH - ändert bestehende Daten auf dem Server
- + DELETE - löscht bestehende Daten auf dem Server
- + ReST wird aus Methoden zusammengesetzt: base URL, URL, media types, etc.



Server - Web - Client



```
// A client request:  
// http://localhost/rest/delete/$xyz1234  
const response = await fetch(`rest/delete/${id}`, {  
  method: 'DELETE',  
  cache: 'no-cache',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
});  
  
// The server acts:  
app.js: app.use('/rest', restRoute);  
  
restRoute: router.delete('/delete/:id?', deletePost);  
  
restController:  
let deletePost = (request, response, next) => {  
  MongoClient.connect(url, (error, dbo) => {  
    if (error) throw error;  
    let database = dbo.db("blog"),  
      myquery = { _id: new ObjectId(request.params.id) };  
  
    database.collection("posts").deleteOne(myquery, (error, resultset) => {  
      if (error) throw error;  
      dbo.close();  
    });  
  });  
  response.status(200).json({ "done": true });
```

[https://www.ibm.com/de-de/cloud/learn/
rest-apis](https://www.ibm.com/de-de/cloud/learn/rest-apis)

[https://jinalshahblog.wordpress.com/
2016/10/06/rest-api-using-node-js-and-
mysql/](https://jinalshahblog.wordpress.com/2016/10/06/rest-api-using-node-js-and-mysql/)

Some ECMAScript 6+

ECMAScript 6 (2015)

1. Let and Const
2. Arrow functions
3. Default parameters
4. for of loop
5. Spread attributes
6. Maps
7. Sets
8. Static methods
9. Classes

Let

```
// let is similar to var but let has scope.  
// let is only accessible in the block level it is  
defined.  
  
if (true) {  
  let a = 40;  
  console.log(a); //40  
}  
  
console.log(a); // undefined
```

Const

// Const is used to assign a constant value to the variable.

// And the value cannot be changed. Its fixed.

```
const a = 50;
```

```
a = 60;           // shows error.
```

You cannot change the value of const.

Arrow Function

```
// Old Syntax
function fn () {
  console.log("Hello World..!");
}

function fn () { . . . }
fn = () => { . . . }

// New Syntax
let fn = () => {
  console.log("Hello World..!");
}

() => { console.log("Hello World..!"); }
```

Arrow Function mit Default

```
let Func = (a, b = 10) => {
  return a + b;
}

Func(20); // 20 + 10 = 30
```

For of loop

```
// for..of iterates through list of elements (i.e) like  
Array  
// and returns the elements (not their index) one by one.  
  
let arr = [2,3,4,1];
```

```
for (let value of arr) {  
  console.log(value);  
}
```

Output:

```
2  
3  
4  
1
```

Maps

```
// Map holds key-value pairs, define an own index. Indexes are unique in maps.

let map = new Map(); //Empty Map
let map = new Map([[1,2],[2,3]]); // map = {1=>2, 2=>3}

map.get(1)// 2
map.has(1);//return boolean value: true/false
map.set(4,5); // {1=>2, 2=>3, 4=>5}

var isDeleteSucceeded = map.delete(1); //{ 2=>3, 4=>5}
console.log(isDeleteSucceeded); //true

map.clear(); // {}

console.log(map.size); //0

//For map: { 2=>3, 4=>5}
for (const item of map){
    console.log(item); //Array[2,3]; //Array[4,5]
}

map.forEach((value, key) => console.log(`key: ${key}, value: ${value}`));
//key: 2, value: 3; //key: 4, value: 5
```

Sets

```
var emptySet = new Set();
var exampleSet = new Set([1,2,3]);

var arr = Array.from(set);//[1,2,3]

console.log(set.has(0)); // boolean - false
console.log(set.has(1)); //true

set.add(3); //{1,2,3}
set.add(4); //{1,2,3,4}

set.delete(4); //{1,2,3}
set.clear(); //{}  

```

Static methods

```
class Example {  
    static Callme() {  
        console.log("Static method");  
    }  
}  
  
Example.Callme();
```

Output:
Static method

Classes

Classes, Klasse als ES Modul

```
class Control {
  constructor(value = 'default') {
    this._myProperty = value;
  }
  get property() {
    return this._myProperty;
  }
  set myProperty(value) {
    value = value.trim();

    if (typeof value === 'undefined') {
      throw new Error('value is undefined');
    }

    this._myProperty = value;
  }
  myMethod = function () {
    ...
  }
}

export default new Control();
```

public
private
~~protected~~

Klassen, Konstruktor,

```
class Shape {  
  
    constructor (id, x, y) {  
        this._id = id;  
    }  
    move (x, y) {  
        this.x = x  
        this.y = y  
    }  
    get id () {}  
    set id () { return this._id; }  
}  
  
let myShape = new Shape('rect', 10, 10);
```

Vererbung, Elternkonstruktor

```
class Rectangle extends Shape {  
    constructor (id, x, y, width, height) {  
        super(id, x, y);  
        this._width = width;  
        this._height = height;  
    }  
}  
  
class Circle extends Shape {  
    constructor (id, x, y, radius) {  
        super(id, x, y);  
        this._radius = radius;  
    }  
}
```

Getter, Setter und underscore-Schreibweise

```
class Shape {  
    constructor () {  
        this.type = "circle"  
    }  
    set type (value) {  
        this._type = value  
    }  
    get type () {  
        return this._type  
    }  
}
```

Webpack

```
npm install webpack webpack-cli --save-dev
npm install clean-webpack-plugin --save-dev
npm install html-webpack-plugin --save-dev
npm install css-loader --save-dev
npm install style-loader --save-dev
npm install babel-loader @babel/core @babel/preset-env webpack

npm install webpack webpack-cli clean-webpack-plugin
css-loader style-loader --save-dev
```

webpack.conf.js

```
/** Webpack Config */

// Import
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-
plugin');

module.exports = {
  mode: 'development', // 'development', 'production'

  // To mix frontend and backend code:
  // instructs webpack to target a specific environment,
  // here 'node'. Defaults to "web", others would be
  'electron', ...
  target: 'node',

  // Instructions, how to build the application
  entry: {
    index: './src/index.js',
    // other: './src/other.js'
  },

  // Instructions, how to export the application
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].js' // index.js, other.js
  },

  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          {
            loader: 'style-loader'
          },
          {
            loader: 'css-loader'
          }
        ],
        // {
        //   test: /\.js$/,
        //   exclude: /node_modules/,
        //   use: [
        //     {
        //       loader: 'babel-loader',
        //       options: {
        //         presets: ['@babel/preset-env']
        //       }
        //     },
        //   ],
        //   // {}, for more loader, i.e. images, fonts,
        //   sass ...
        // },
        // plugins: [
        //   new CleanWebpackPlugin(),
        //   new HtmlWebpackPlugin({
        //     template: './src/index.html',
        //     filename: 'index.html'
        //   })
        // ]
      }
    ]
  }
}
```

Datenaustausch und Kommunikation zwischen Server und Client

fetch()

AJAX - ASYNCRONOUS JAVASCRIPT AND XML

- + AJAX ist eine standardisierte Methode, um per Javascript Daten vom Server anzufordern.
- + AJAX basiert auf dem **XMLHttpRequest** Object.

```
let request = new XMLHttpRequest();
request.open(filename, 'get');
request.send();
-----
request.addEventListener('readystatechange',
function(){ ... });
```

AJAX - ASYNCRONOUS JAVASCRIPT AND XML

```
 xhr = new XMLHttpRequest();
xhr.addEventListener('readystatechange', onReadyStateChange);
xhr.open("GET", _file);
xhr.send();

function onReadyStateChange() {
    switch (xhr.readyState) {
        case 0:
            console.log('there is no request');
            break;
        case 1:
            console.log('request opened');
            break;
        case 2:
            console.log('request sent');
            break;
        case 3:
            console.log('response first part ...');
            break;
        case 4:
            console.log('response more parts and finished!');
            if (xhr.status === 200 || xhr.status === 304) {
                _data = JSON.parse(xhr.response);
            }
            break;
        default:
            console.log('something strange happened!');
            break;
    }
}
```

fetch()(ES 6+)

```
fetch('data.json')
  .then(function (response) {
    if (response.ok)
      return response.json();
    else
      throw new Error('Kurse konnten nicht geladen werden');
  })
  .then(function (json) {
    // Hier Code zum einarbeiten der Kurse in die Anzeige
  })
  .catch(function (err) {
    // Hier Fehlerbehandlung
  });

// fetch basiert auf Promises,
// die ebenfalls in ES 6+ eingeführt wurden
```

fetch() einstellen

```
fetch( url, {  
  method: 'POST',                      // *GET, POST, PUT, DELETE, etc.  
  mode: 'cors',                        // no-cors, *cors, same-origin  
  cache: 'no-cache',                   // *default, no-cache, reload, force-cache,  
                                      // only-if-cached  
  credentials: 'same-origin',          // include, *same-origin, omit  
  headers: {  
    'Content-Type': 'application/json' // 'application/x-www-form-urlencoded',  
  },  
  redirect: 'follow',                  // manual, *follow, error  
  referrerPolicy: 'no-referrer',       // no-referrer, *no-referrer-when-downgrade,  
                                      // origin, origin-when-cross-origin,  
                                      // same-origin, strict-origin,  
                                      // strict-origin-when-cross-origin,  
                                      // unsafe-url  
  body: JSON.stringify(data)           // body data type must match "Content-Type" header  
})
```

[https://developer.mozilla.org/en-US/docs/
Web/API/Fetch_API/Using_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)

[https://dmitripavlutin.com/javascript-
fetch-async-await/](https://dmitripavlutin.com/javascript-fetch-async-await/)

Mehr über `fetch()`

Axios

SOLID

SOLID - die Wartbarkeit von Software verbessern

- + Single Responsibility
- + Open-Closed
- + Liskovsche Substitution
- + Interface Segregation
- + Dependency Inversion

„S“ wie „Single-Responsibility-Prinzip“

- + „Es sollte nie mehr als einen Grund dafür geben, eine Klasse zu ändern.“ Robert C. Martin
- + Jede Klasse hat nur genau eine fest definierte Aufgabe zu erfüllen. Wenn eine Klasse mehrere Verantwortungen zu tragen hat, führt das zu Schwierigkeiten bei zukünftigen Änderungen und das Fehlerrisiko steigt.
- + Eine hohe Kohäsion wird angestrebt. Alle Methoden innerhalb einer Klassen haben einen starken gemeinsamen Bezug
- +

„O“ wie „Open-Closed-Prinzip“

- + Module sollten offen für Erweiterungen sein, aber geschlossen für Modifikationen.
- + Klassen, Methoden, Module werden so entwickelt, dass sie einfach zu erweitern sind – ohne ihr Verhalten zu ändern.
- + Beispiel Vererbung: Das Verhalten einer Klasse wird nicht verändert, erhöht aber trotzdem die Funktionalität der Software.
- + Auch das Überschreiben von Methoden verändert nicht das Verhalten der Basisklasse, sondern nur die Methoden der abgeleiteten Klasse.

„L“ wie „Liskovsches Substitutionsprinzip“

- + „Sei $q(x)$ eine Eigenschaft des Objektes x vom Typ T , dann sollte $q(y)$ für alle Objekte y des Typs S gelten, wobei S ein Subtyp von T ist.“ Barbara Liskov

„I“ wie „Interface-Segregation-Prinzip“

- + „Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden.“ Robert C. Martin
- + Wie der Name erahnen lässt, geht es hierbei darum, Interfaces aufzuspalten beziehungsweise sie nicht unnötig groß zu machen.
- + Ein Interface soll also so gestaltetet sein, dass die Anforderungen des Clients erfüllt werden können. Damit soll vermieden werden, dass ein Client mehrere Interfaces nutzen muss, die er für seine eigentliche Anforderung gar nicht benötigt werden.
- +

„D“ wie „Dependency-Inversion-Prinzip“

- + A. Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen.
- + B. Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.“ Robert C. Martin
- + Das Prinzip beschäftigt sich mit der Abhängigkeit von abgeschlossenen funktionalen Einheiten einer Software – also Modulen. In Modulen, die eine höhere Hierarchie innerhalb der Software aufweisen, werden generelle Abläufe beschrieben, die von spezielleren Modulen verwendet werden.
- + Je niedriger das Modul innerhalb der Hierarchie, desto spezifischere Probleme löst es.
- + Wenn ein hierarchisch niedriges Modul von einem höherliegendem Modul abhängig ist, entsteht ein Problem. Änderungen in spezifischen Modulen ändern somit das Verhalten von höher liegenden und generellen Modulen, eine zyklische Abhängigkeit entsteht und die Architektur und das Design der Software werden unnötig komplex. Das Prinzip soll also eine Invertierung der Abhängigkeiten sicherstellen.
- +