



WILLKOMMEN ZU  
NODEJS GRUNDLAGEN

MICHAEL REICHART, DEZEMBER 2021

# 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

Dahl was inspired to create Node.js after seeing a file upload progress bar on Flickr. The browser did not know how much of the file had been uploaded and had to query the web server. Dahl wanted an easier way.

– 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

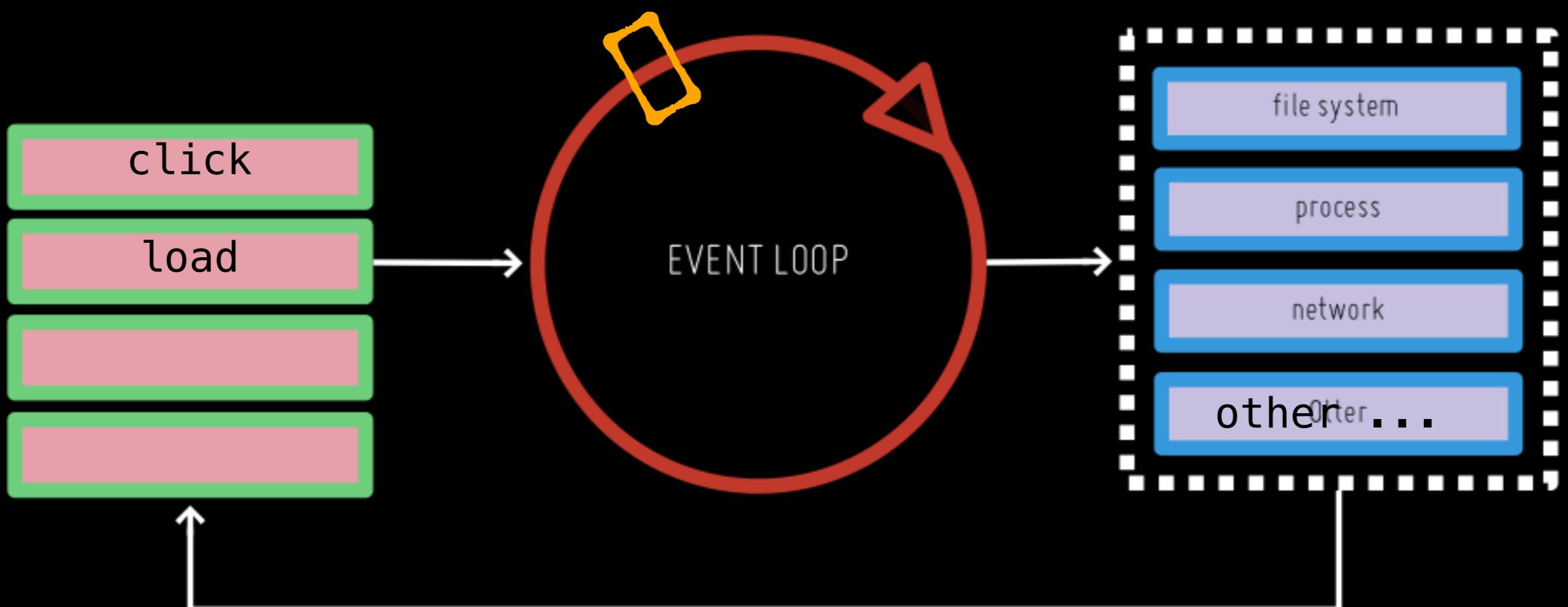
- Client und Server technisch auf derselben Basis stehen.
- Format- und Kompatibilitätsprobleme zwischen unterschiedlichen Sprachen, zum Beispiel Javascript und PHP wegfallen.
- JavaScript alle Aufgaben eines Web-, Socket- oder Datenservers übernehmen kann.

# JAVASCRIPT AUF DEM SERVER ZU VERWENDEN, IST GUT, WEIL

- Server und Client zu einem Ganzen verschmelzen.

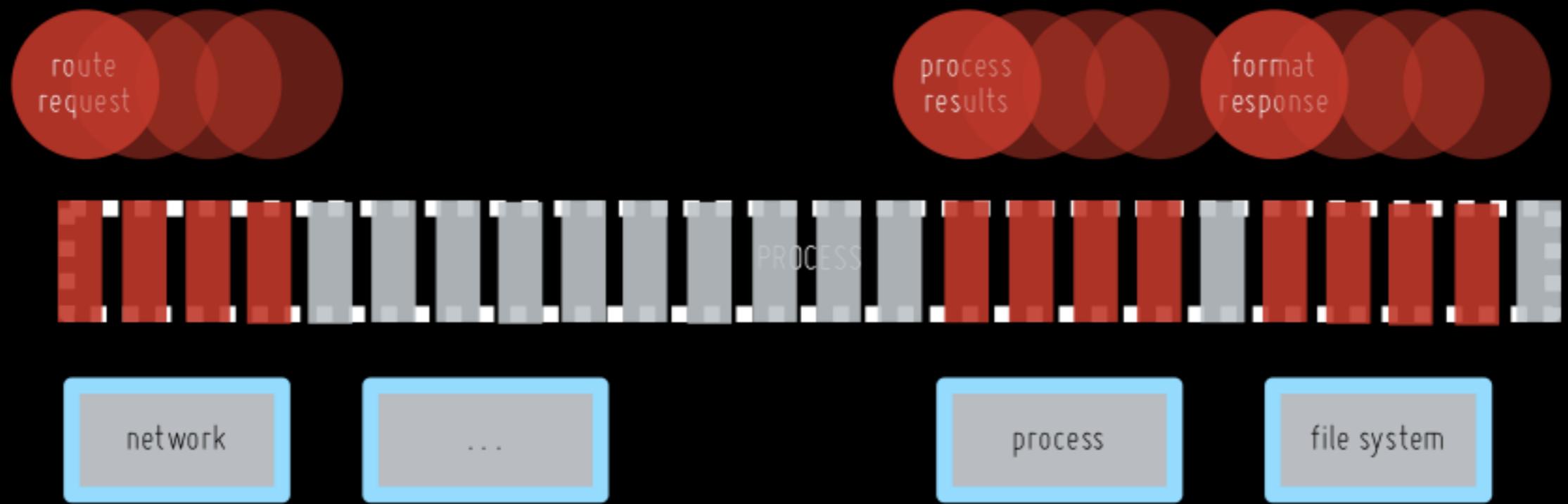
# NON BLOCKING I/O, EVENT LOOPS, SINGLE THREADS JAVASCRIPT THREAD VERHALTEN UND EVENTLOOP

EVENT QUEUE  
(EVENTS AND CALLBACKS)

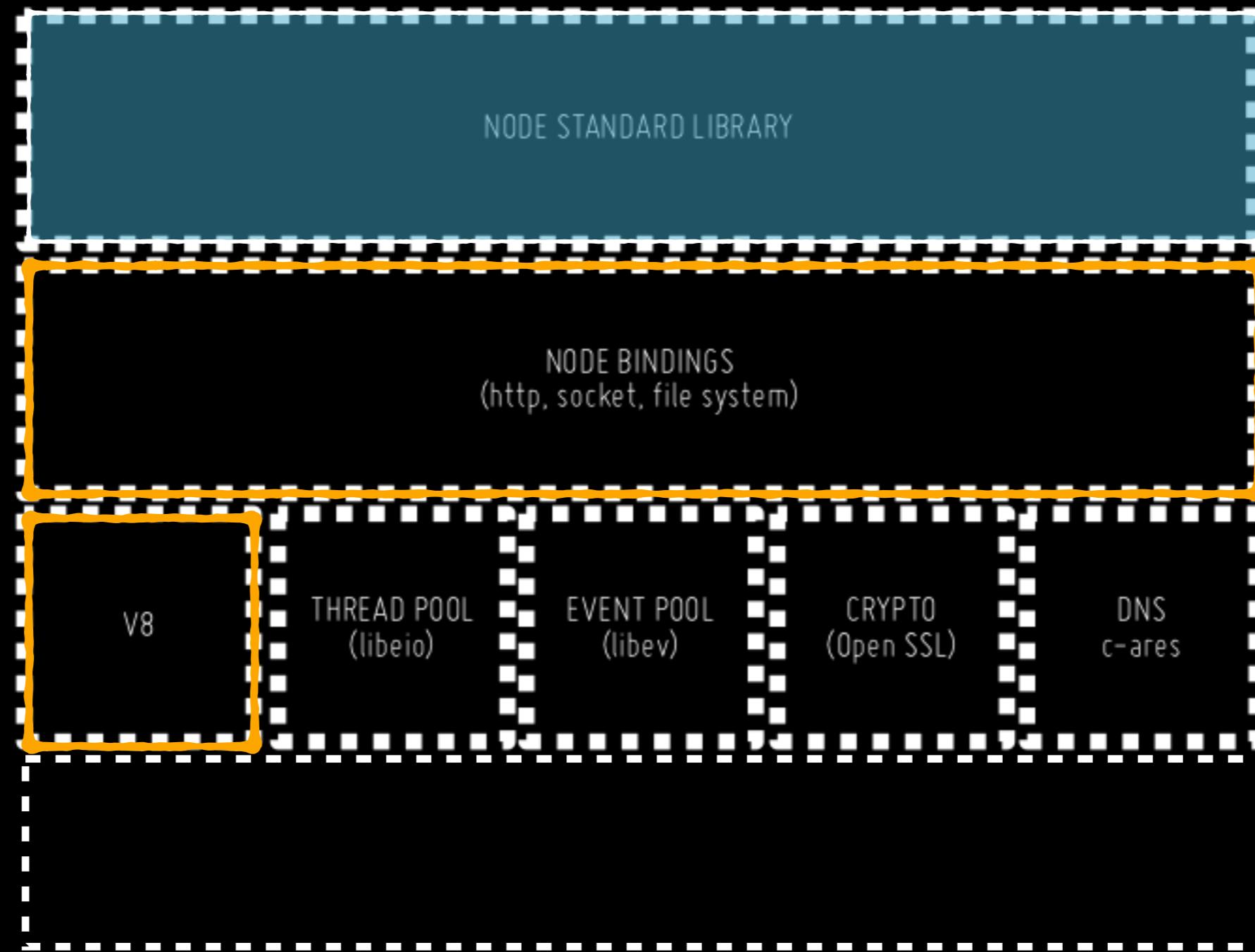


THREAD POOL

## SCALING WITH AN EVENT LOOP



## NODE TIERS AND BINDINGS



# FROM THE SCRATCH 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.

## Download for Windows, Linux, Mac OS

**14.17.0 LTS**

Recommended For Most Users

**16.3.0 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

## Willkommen bei: Node.js

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



**This package will install Node.js v16.13.1 and npm v8.1.2 into /usr/local/.**

Zurück

Fortfahren

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.



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

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

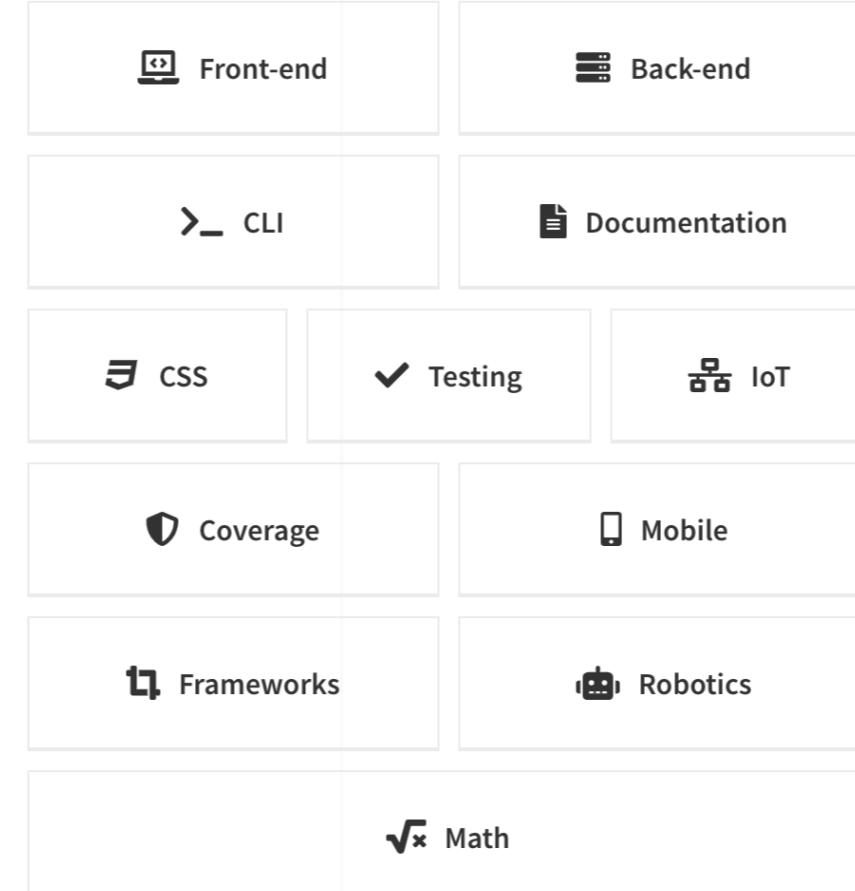
commander

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

## 3 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);
```

# HALLO WELT - KONSOLENVERSION

```
$ node server
```

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  
...
```

# 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>

# AUSGABEN ÜBER DIE KONSOLE

# KONSOLENBEFEHLE

```
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.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

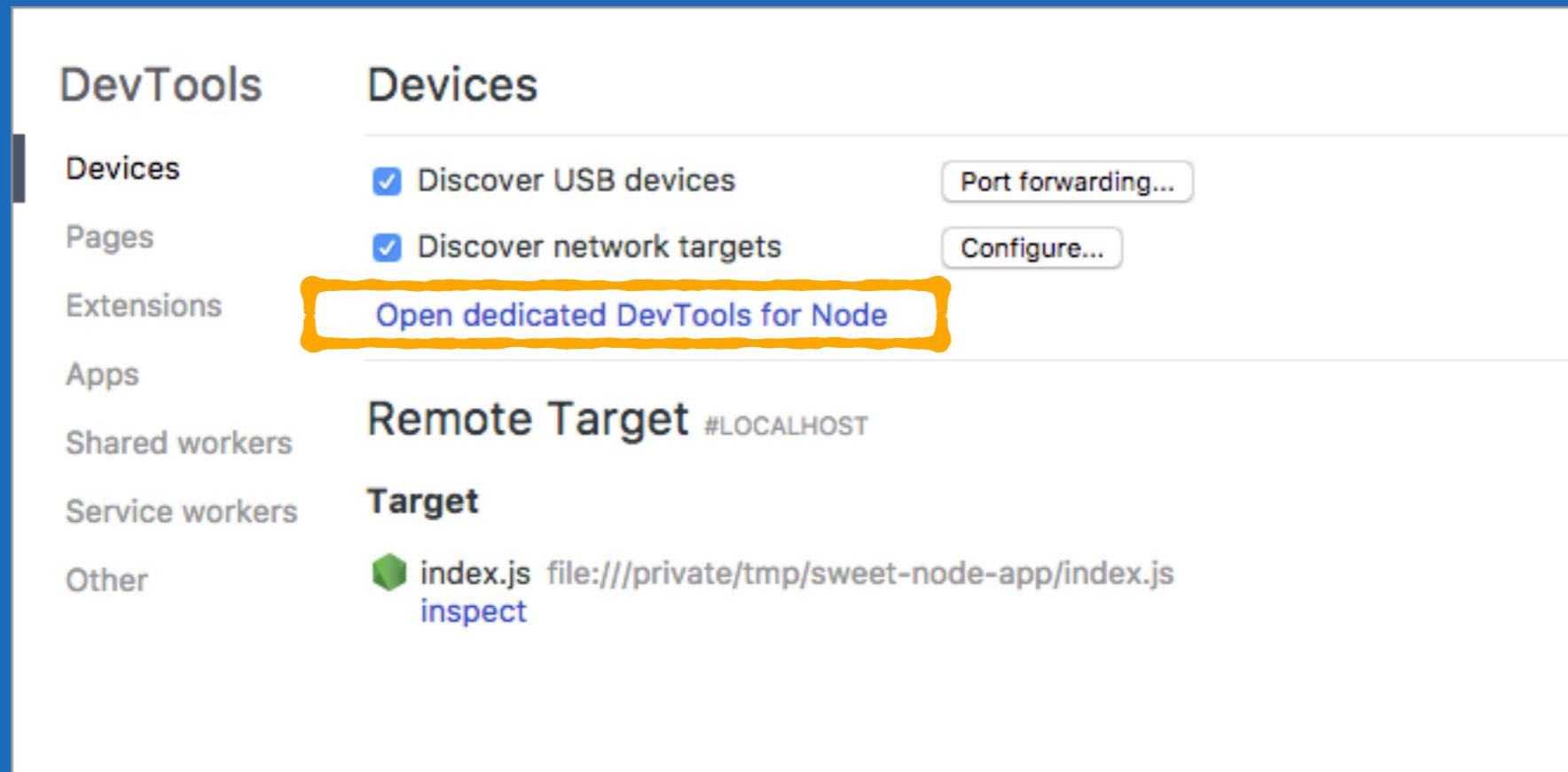
- |   |  |
|---|--|
| <code>setBreakpoint(),<br/>sb()</code>                    | – Set breakpoint on current line                           |
| <code>setBreakpoint(line),<br/>sb(line)</code>            | – Set breakpoint on specific line                          |
| <code>setBreakpoint('fn()'),<br/>sb(...)</code>           | – Set breakpoint on a first statement<br>in functions body |
| <code>setBreakpoint('script.js', 1),<br/>sb(...)</code>   | – Set breakpoint on first line of script.js                |
| <code>clearBreakpoint('index.js', 15),<br/>cb(...)</code> | – Clear breakpoint in index.js on line 15                  |

# DEBUGGING MIT DEN CHROME DEVTOOLS

# DER NODEJS V8 INSPECTOR

- Die Chrome Extension "nodejs V8 inspector" in einer aktuellen Chromeverision installieren.
- `node --inspect index.js`
- `# Break on the first statement of the script with --inspect-brk`
- `node --inspect-brk index.js`
- startet einen Port **9229**, über den die Chrome Devtools Fehlermeldungen anzeigen und der Code fehlerbehandelt werden kann.

- chrome://inspect



# EIN NODE PROJEKT AUFSETZEN

# NODE PROJEKT

## **npm init**

npm then asks some questions and builds a package.json file.  
Then start building the project.

# NODE PROJEKT

**npm init -y**

accepts all of the default options that npm init asks you about

# NODE PROJEKT

## **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

# CUSTOMIZING NODE INIT

See the init vectors

```
npm config list | grep init
```

```
npm set init.author.name "Your name"
```

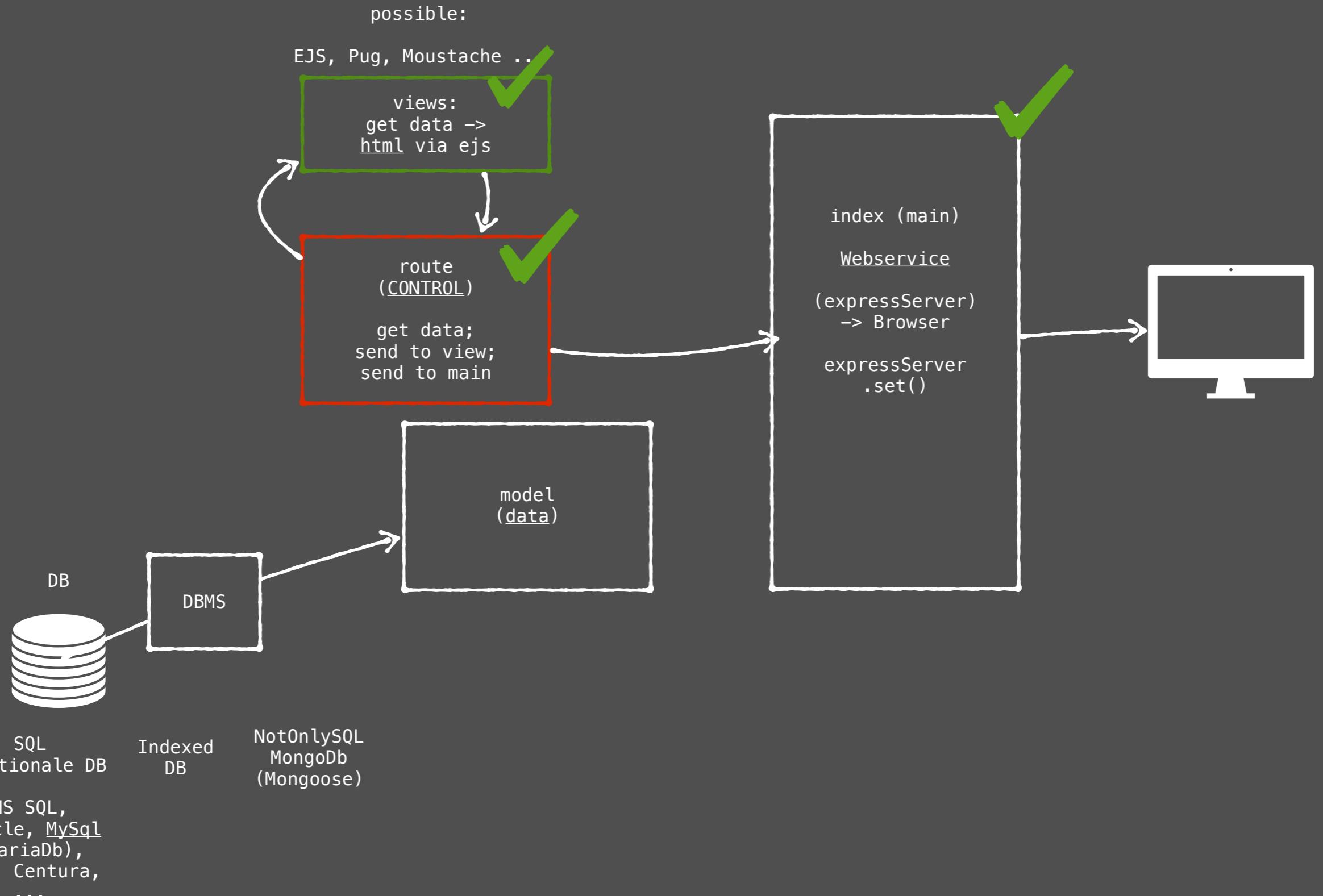
```
npm set init.author.email "your@email.com"
```

```
npm set init.author.url "https://your-url.com"
```

```
npm set init.license "MIT"
```

```
npm set init.version "1.0.0"
```

Once you have that customised to your liking, `npm init -y` will always produce the right settings.



GET → READ DATA  
 POST → CREATE DATA  
 PATCH → UPDATE DATA  
 DELETE → DELETE DATA

2021: ReST API?  
JSON, HTTP  
Schnittstelle für Systeme <→> Daten

2003: SOAP (XML)

```

index (main)
Wbservice

routes:

GET "/user/:id/"
"SELECT id FROM user"
-> response.json()

POST "/user/23/"
request.body: {user.data}
"INSERT INTO user ..."

PATCH "/user/23/"
request.body: {user.data}
"UPDATE user WHERE id=23"

DELETE "/user/23/"
"DELETE FROM user WHERE id=23"
  
```



SQL  
Relationale DB  
MS SQL,  
Oracle, MySql  
(MariaDb),  
DB2, Centura,  
...

Indexed DB

NotOnlySQL  
MongoDb  
(Mongoose)



file.watch()



URI  
http://REQUEST



json

ws://server-message

http://RESPONSE

JSON



# EIN MODUL SCHREIBEN

# MODUL SCOPES

- Eine Javascript Datei bildet einen eigenen Scope.
- Auf ein IIFE Pattern kann verzichtet werden.
- require() und exports verbinden Prozess und Module.

# EIN NODE MODUL SCHREIBEN

```
let
  _area = null,
  _circumference = null;

_area = function(radius) {
  'use strict';
  return Math.PI * radius * radius;
};

_circumference = function(radius) {
  'use strict';
  return Math.PI * 2 * radius;
};

let private = 'geheim';

export default {
  area: _area,
  circumference: _circumference
}

// module.exports.area = _area;
// module.exports.circumference = _circumference;
```

# EIN MODUL VERWENDEN

```
import module from './module.js';
// myModule = require('./module.js');

let radius = 5;

console.log(module.circumference(radius[i]));
console.log(module.area(radius[i]));
```

# MODULE VERWALTEN

# PACKAGE.JSON

```
{  
  "name" : "my-express-app",    // < 215 Zeichen, keine Großbuchstaben, url-safe  
  "version": "0.0.1",  
  "description" : "Maecenas sed diam eget risus varius blandit.",  
  "license": "MIT",  
  "repository": {  
    "type" : "git",  
    "url"  : "git+https://github.com/zenbox/workshop.git"  
  },  
  "contributors": [  
    {  
      "name": "Michael Reichart",  
      "email": "michael@zenbox.de"  
    }  
  ],  
  "dependencies" : {  
    "express" : "4.15.2",  
    "ejs" : "2.5.6",  
    "jade" : "1.11.0",  
    "body-parser" : "1.17.1",  
    "cookie-parser": "1.4.3",  
    "socket.io": "1.7.3"  
  }  
}
```

# EIN MODULE INSTALLIEREN

```
$ npm install
```

# 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');
```

# 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.x",  
    "modulename" : "1.2.*",  
    "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  
  }  
}
```

"<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');
  }
});
```

# MySQL

# THE DATABASE CONNECTION DATA

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

  "more": {
    "database": "application",
    "table": "user"
  }
}
```

# A DATABASE CONNECTION

```
'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();
```

# 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>
```

# DIE UTILITIES KLASSE

```
util.debuglog(section)
util.deprecate(function, string)
util.format(format[, ...args])
util.inherits(ctor, superCtor)
util.inspect(obj[, 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);  
});
```

# DER 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);
```

# ECHTZEITKOMMUNIKATION ZWISCHEN CLIENT UND SERVER WEB SOCKETS

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

– JAN HICKSON

# HTTP IST STATUSLOS

- -> Request, Response, Ende
- Echtzeit ist nicht möglich, da Client und Server bei jedem Request erneut verhandeln, bevor Daten übertragen werden.

# ECHTZEITVERBINDUNGEN?

- Seiten aktualisieren
- Polling
- Longpolling

- Polling und Longpolling setzen dabei in regelmäßigen Intervallen Requests an den Server ab, um neue Informationen zu erhalten.
- Beim Longpolling hält der Server die Verbindung eine Zeit lang offen.

- Keine echte Synchronisation mit severseitigen Informationsupdates

# 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 Verbindungsauflauf einer WebSocket-Verbindung läuft über HTTP (oder HTTPS),
- Ein Upgrade-Header teilt dem Server mit, dass auf das WebSocket-Protokoll „upgegradet“ werden soll.
-

```
// From client to server:  
GET /chat HTTP/1.1  
Host: server.example.com  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Key: dGhIHNhbXBsZSSub25jZQ==  
Origin: http://example.com  
Sec-WebSocket-Protocol: chat, superchat  
Sec-WebSocket-Version: 13  
  
// From server to client:  
HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzhZRbK+x0o=  
Sec-WebSocket-Protocol: chat
```

# SERVER PROXY

- Serverseitig wird ein Socketproxy installiert, zum Beispiel in node.js.
- nodejs beherrscht HTTP und Websockets und kann einen Upgrade-Headers verarbeiten.

# WEITERE SOCKET SERVER

- **Jetty**  
HTTP-Server und Servlet-Container, der WebSockets seit Version 7.0.1 unterstützt.
- **phpwebsocket**  
einer der ersten WebSocket-Server in PHP
- **jWebSocket**  
High-Speed Kommunikationsserver inkl. Web, Java und Mobile Clients, Open Source
- **xLightweb**  
HTTP-Bibliothek inkl. HttpClient und HttpServer, welche WebSocket und ServerSentEvent unterstützt

# SOCKET TOOLS

- **Tomcat**  
ab Version 7.0.27
- **PyWebsocket**  
Python Websocket Server (Apache Modul oder Standalone)
- **Node.js**  
Serverseitiges Javascript, mit dem ein Webserver geschrieben werden kann
- **Socket.io**  
Ein Framework, mit dem realtime apps für jeden Browser und jedes mobile Gerät möglich sind.

# HTML5 SOCKETS BROWSER

- Chrome 4+
- Firefox 4+
- Safari 5+
- IE 10+
- Opera 11.5+

# WAS BRINGEN WEB SOCKETS?

# VORTEILE DER WEBSOCKETS GEGENÜBER HTTP REQUESTS

- Geschwindigkeitsteigerung
- Datenmengenreduktion

# 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
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
```

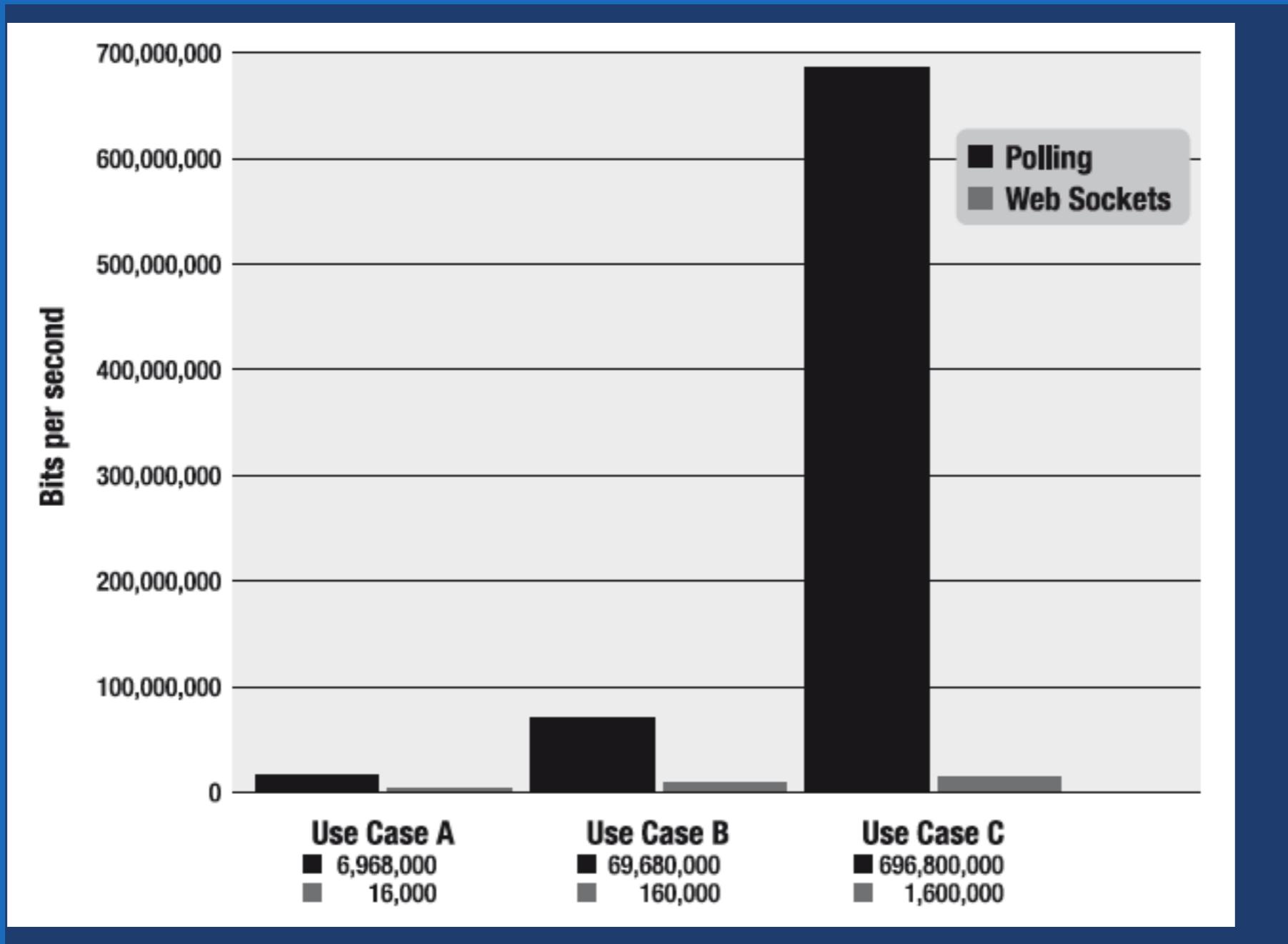
- 1,000 Clients pollen jede Sekunde  
6,968,000 bits per second  
6.6 Mbps
- 10,000 Clients pollen jede Sekunde  
69,680,000 bits per second  
66 Mbps
- 100,000 Clients pollen jede Sekunde  
696,800,000 bits per second  
665 Mbps

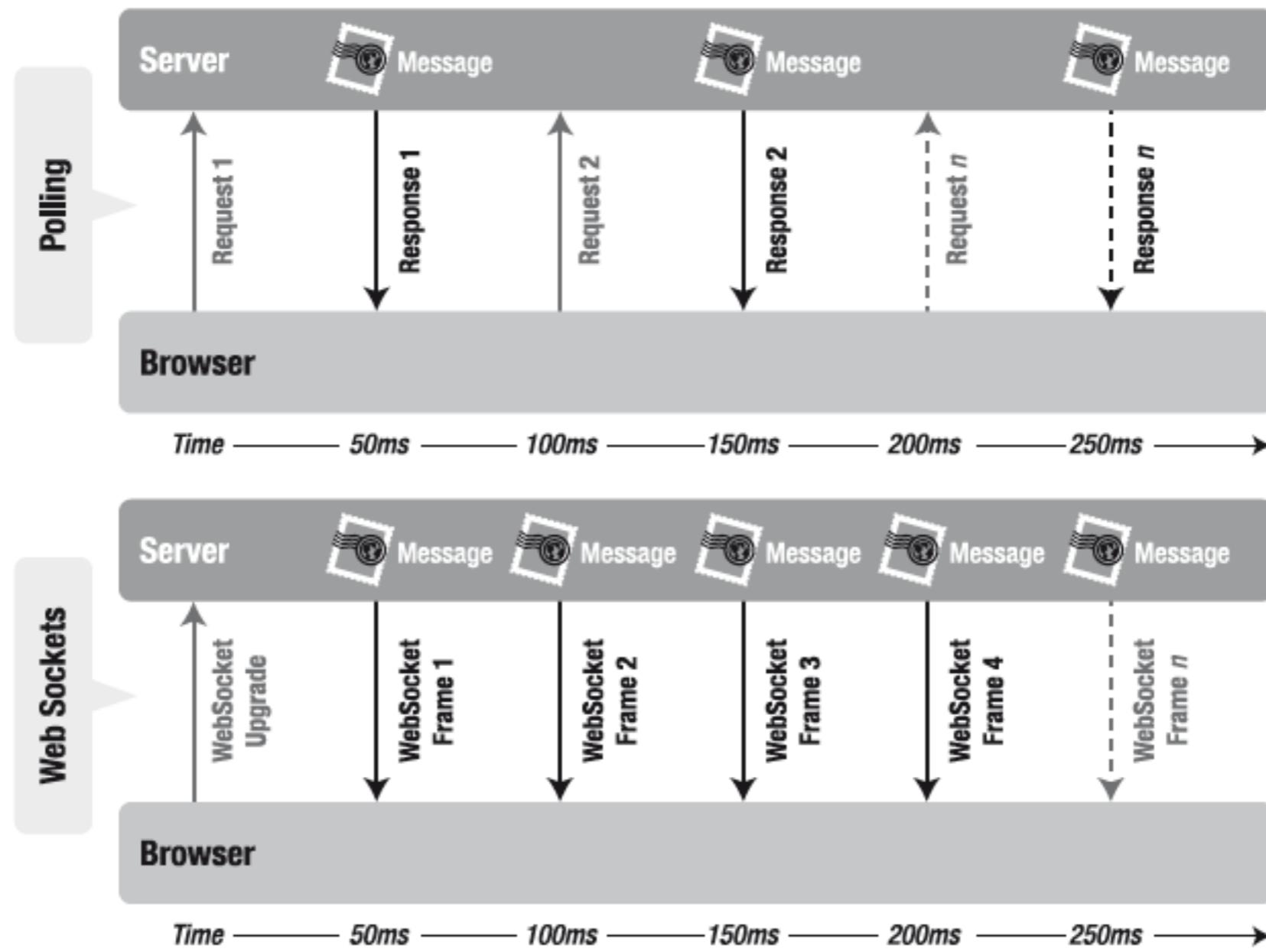
# HEAD EINES SOCKET-REQUEST

\0x00 Hello, WebSocket \0xff

0,002 %

- 1,000 Clients erhalten 1 Nachricht pro Sekunde  
16,000 bits per second  
0,015 Mbps vs. 6.6 Mbps
- 10,000 Clients erhalten 1 Nachricht pro Sekunde  
160,000 bits per second  
0,153 Mbps vs. 66 Mbps ->
- 100,000 Clients erhalten 1 Nachricht pro Sekunde  
1,600,000 bits per second  
1,526 Mbps vs. 665 Mbps





# DAS WEBSOCKET INTERFACE

```
// The Websocket Interface – start und ready state section

[Constructor(DOMString url, optional (DOMString or DOMString[])
protocols)] interface WebSocket : EventTarget {
    readonly attribute DOMString url;

    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;

    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;
    ...
}
```

```
// The Websocket Interface – networking

[TreatNonCallableAsNull] attribute Function? onopen;
[TreatNonCallableAsNull] attribute Function? onerror;
[TreatNonCallableAsNull] attribute Function? onclose;

readonly attribute DOMString extensions;
readonly attribute DOMString protocol;

void close([Clamp] optional unsigned short code,
          optional DOMString reason);
...
```

```
//The Websocket Interface – messaging and end section

[TreatNonCallableAsNull]
attribute Function onmessage;
attribute DOMString binaryType;

void send(DOMString data);
void send(ArrayBufferView data);
void send(Blob data);
};
```

Quelle: <http://dev.w3.org/html5/websockets/>; Hickson

# 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();  
    }  
};
```

# JAVASCRIPT AUF DEM SERVER 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“);  
}
```

# LIVE RELOAD

# LIVE RELOAD IN BROWSER

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

```
// use with express
app.use(require('connect-livereload')({
  port: 35729
}));
```

```
@see https://www.npmjs.com/package/connect-livereload
```

# CRYPTO

# CIPHER / 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();
```

# CIPHER / 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);
```

# CLIENT-SERVER-MODELL

- REST verlangt ein Client-Server-Modell, will also das Nutzerinterface von der Datenhaltung getrennt sehen.
- Damit sollen sich Clients einerseits leichter auf verschiedenen Plattformen portieren lassen;
- vereinfachte Serverkomponenten sollen andererseits besonders gut skalieren.

# ZUSTANDSLOSIGKEIT

- Client und Server müssen zustandslos („stateless“) miteinander kommunizieren. Das bedeutet: Jede Anfrage eines Clients beinhaltet alle Informationen, die ein Server benötigt;
- Server selbst können auf keinen gespeicherten Kontext zurückgreifen. Dieses Constraint verbessert damit Visibility, Zuverlässigkeit und Skalierbarkeit.
- Hierfür nimmt REST jedoch Nachteile bei der Netzwerkperformanz in Kauf;
- überdies verlieren Server die 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 Informationen müssen dem entsprechend als „cacheable“ oder „non-cacheable“ gekennzeichnet werden.
- Die Vorteile responsiverer Anwendungen mit höherer Effizienz und Skalierbarkeit werden dabei mit dem Risiko erkauft, dass Clients auf veraltete Daten aus dem Cache zurückgreifen.

# EINHEITLICHE SCHNITTSTELLE

- Die Komponenten REST-konformer Services nutzen eine einheitliche, allgemeine und vom implementierten Dienst entkoppelte Schnittstelle.
- Ziel des Ganzen sind eine vereinfachte Architektur und eine erhöhte Visibilität von Interaktionen. Dafür nimmt man eine schlechtere Effizienz in Kauf, wenn Informationen in ein standardisiertes Format gebracht – und nicht für die Bedürfnisse spezieller Anwendungen angepasst – 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 überdies 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.

# 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

```
class MyClass {  
    constructor () {}  
  
    getProperty() {}  
    setProperty() {}  
}
```

~~public  
private  
protected~~

# CLASSES

```
class ChildClass extends parentClass {  
    constructor () {}  
  
    getProperty() {}  
    setProperty() {}  
}
```

# KLASSEN, KONSTRUKTOR,

```
class Shape {  
  
    constructor (id, x, y) {  
        this.id = id. // property!  
        this.move(x, y)  
    }  
    move (x, y) {  
        this.x = x  
        this.y = y  
    }  
    getId () {}  
    setMove(){}  
}  
  
let myShape = new Shape('rect', 10, 10);
```

```
ECMAScript 5 – syntactic sugar: reduced | traditional  
var Shape = function (id, x, y) {  
    this.id = id;  
    this.move(x, y);  
};  
Shape.prototype.move = function (x, y) {  
    this.x = x;  
    this.y = y;  
};
```

# 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;  
    }  
}
```

# TYPESCRIPT IN NODEJS

# TYPESCRIPT

- ein typisiertes "Superset" für Javascript, das nach Javascript transpiliert wird.
- **optionale** statische Typisierung. JavaScript Apps lassen sich daher einfach portieren.
- Unterstützt ECMAScript Features, die von aktuellen V8 Versionen noch nicht unterstützt werden.  
(siehe auch <https://node.green/>)
- Bessere OO: Klassen, aber auch Interfaces
- Bessere Zusammenarbeit mit IntelliSense oder Visual Studio Code.

# TYPISIERUNG

- String haben eine `toLowerCase()` method, aber keine `parseInt()` Methode.
- Erweiterbare Typen: Eigene Typen können definiert werden.
- Aus `.js` wird `.ts`.  
Dann ist eine schrittweise Einführung in die Arbeit möglich.

# TYPESCRIPT INSTALLIEREN

```
$ npm install -g typescript  
$ tsc --init
```

# EIN ERSTES TYPESCRIPT

```
// helloPerson.ts

function sayHelloTo(person: string) {
    return `Hello ${person}!
}

const myName = 'Michael'

console.log(sayHelloTo(myName))
```

# EINE TYPESCRIPT APP

```
project
|- dist
|- src
|   |- App.spec.ts
|   |- App.ts
|   |- index.ts
|
|- node_modules
|- .eslintrc.yaml
|- .gitignore
|- Dockerfile
|- package.lock.json
|- package.json
|- README.md
|- tsconfig.json
```

# APP.TS

```
import * as express from 'express' // ES6!

class App {
    public express

    constructor () {
        this.express = express()
        this.mountRoutes()
    }

    private mountRoutes (): void {
        const router = express.Router()
        router.get('/', (request, response) => {
            response.json({
                message: 'Hello World!'
            })
        })
        this.express.use('/', router)
    }
}

export default new App().express
```

# INDEX.TS

```
import app from './App'

const port = process.env.PORT || 3000

app.listen(port, (err) => { // arrow function syntax!
  if (err) {
    return console.log(err)
  }
  return console.log(`server is now listening on ${port}`)
})
```

# TSConfig.json - Compiler Konfiguration

```
{  
  "compilerOptions": {  
    "target": "es6",  
    "module": "commonjs",  
    "outDir": "dist",  
    "sourceMap": true  
  },  
  "files": [  
    "./node_modules/@types/mocha/index.d.ts",  
    "./node_modules/@types/node/index.d.ts"  
  ],  
  "include": [  
    "src/**/*.ts"  
  ],  
  "exclude": [  
    "node_modules"  
  ]  
}
```

# TYPESCRIPT

# TYPISIERTE VARIABLEN

```
let isDone: boolean = false;

let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;

let color: string = "blue";
let fullName: string = `Bob Bobbington`;
let sentence: string = `Hello, my name is ${ fullName }.

let list: number[] = [1, 2, 3];
let list: Array<number> = [1, 2, 3];
```