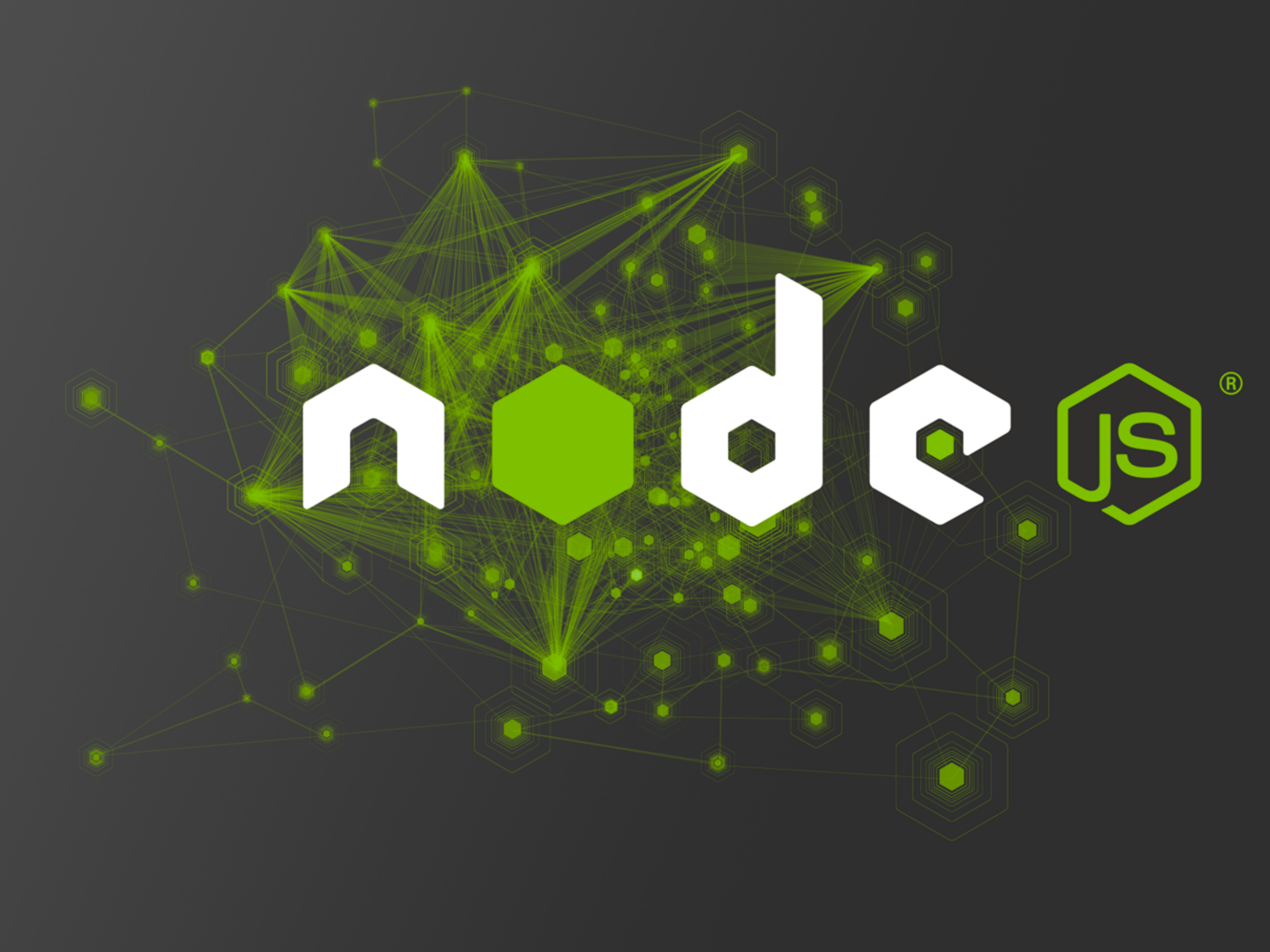
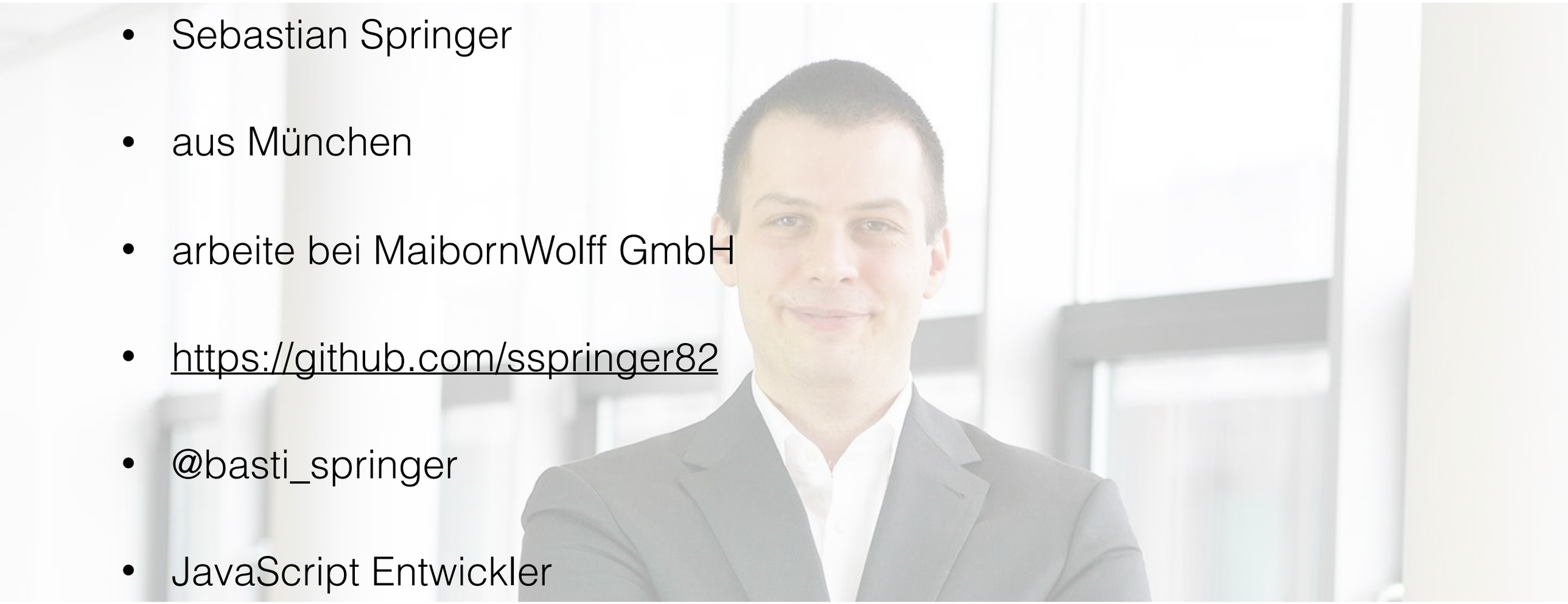

node
JS

Maiborn Wolff
Mensch IT

Basti



- Sebastian Springer
- aus München
- arbeite bei MaibornWolff GmbH
- https://github.com/sspringer82
- @basti_springer
- JavaScript Entwickler

Repo

https://github.com/sspringer82/qs

Setup

Downloads

Latest LTS Version: **v6.9.4** (includes npm 3.10.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Windows Installer	Macintosh Installer	Source Code
Current Latest Features	node-v6.9.4-x86.msi	node-v6.9.4.pkg	node-v6.9.4.tar.gz

Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.exe)	32-bit	64-bit	
macOS Installer (.pkg)	64-bit		
macOS Binaries (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit	64-bit	
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v6.9.4.tar.gz		

Hilfreiche Links

https://nodejs.org/api/index.html

https://developer.mozilla.org

https://github.com/airbnb/javascript

Ausführung

Node kann **interaktiv** auf der Kommandozeile betrieben werden (**REPL**). Oder im regulären **Applikationsbetrieb** mit einer **Datei** als Übergabe.

REPL

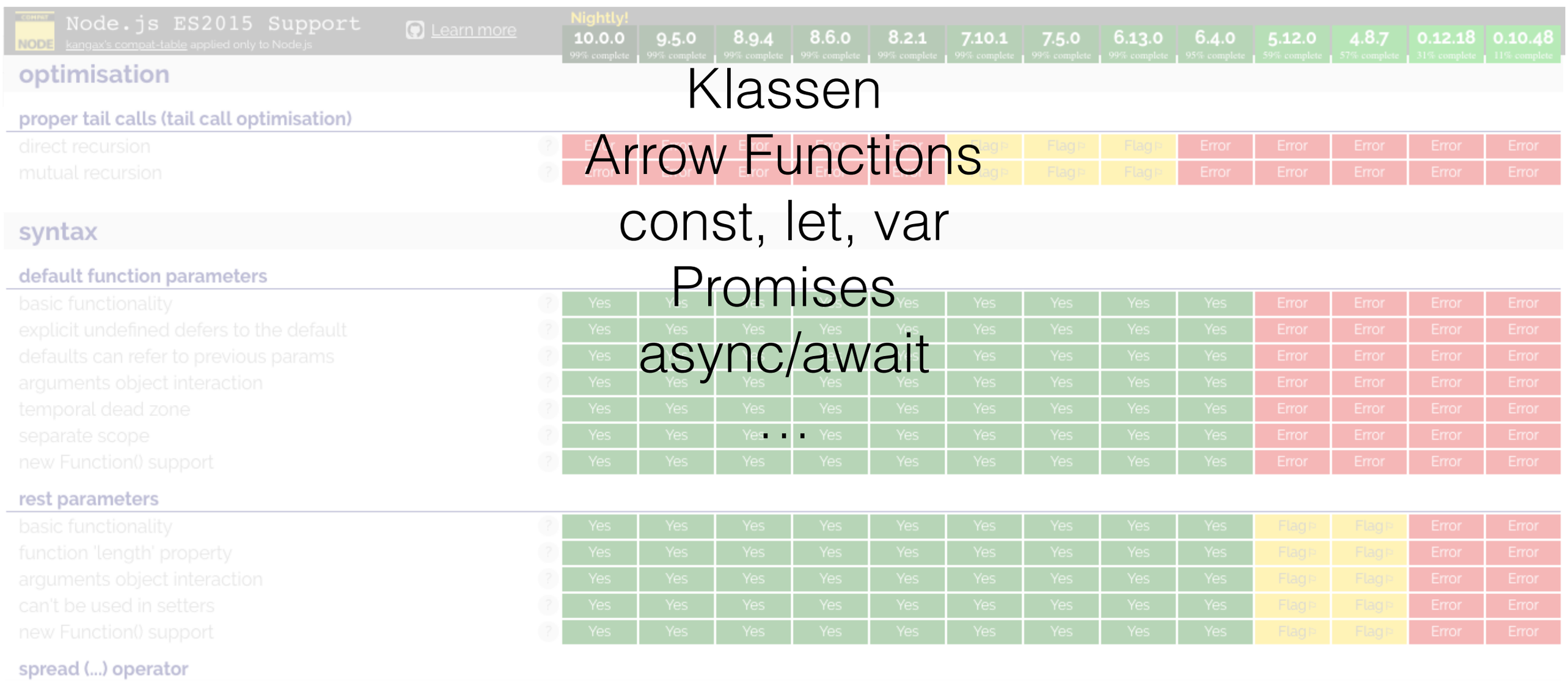
Kommando	Bedeutung
.break	Mehrzeilen-Statement abbrechen
.clear	Aktuellen Kontext zurücksetzen
.exit	REPL beenden
.help	Kommandoliste anzeigen
.save	Aktuelle Session speichern: .save ./file/to/save
.load	Gespeicherte Session laden: .load ./file/to/load
.editor	Editor Session beginnen

Hello World

```
const name = 'World';
const greeting = 'Hello';

console.log(greeting + ' ' + name);
```

Unterstützte Syntax

http://node.green



Klassen

Arrow Functions

const, let, var

Promises

async/await

...

Was	Unix	Windows
node	/usr/local/bin	C:\Program Files\nodejs
npm	/usr/local/bin	C:\Program Files\nodejs
node_modules	/usr/local/lib	C:\Program Files\nodejs

Releases

Release	LTS Status	Codename	Active LTS Start	Maintenance Start	Maintenance End
v0.10.x	**End-of-Life**	-	-	2015-10-01	2016-10-31
v0.12.x	**End-of-Life**	-	-	2016-04-01	2016-12-31
4.x	**Maintenance**	Argon	2015-10-01	2017-04-01	April 2018
5.x	No LTS				
6.x	**Active**	Boron	2016-10-18	April 2018	April 2019
7.x	No LTS				
8.x	**Pending**	Carbon	October 2017	April 2019	December 2019[2]
9.x	No LTS				
10.x	**Pending**	Pending	October 2018	April 2020	April 2021

NVM

NVM

Der **Node Version Manager** ist ein Werkzeug, der es erlaubt **mehrere Versionen** von Node.js parallel auf einem System zu installieren.

NVM ist ein **Bash-Script**, das die verschiedenen Versionen verwaltet.

Die **Installation** erfolgt über ein Script: curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash

Alternativen zu NVM auf **Windows** sind **nvm-windows** oder **nodist**

NVM

nvm install <version>	Installiert die angegebene Version
nvm use <version>	Schaltet zur angegebenen Version um
nvm list	Zeigt eine Liste der verfügbaren Versionen an

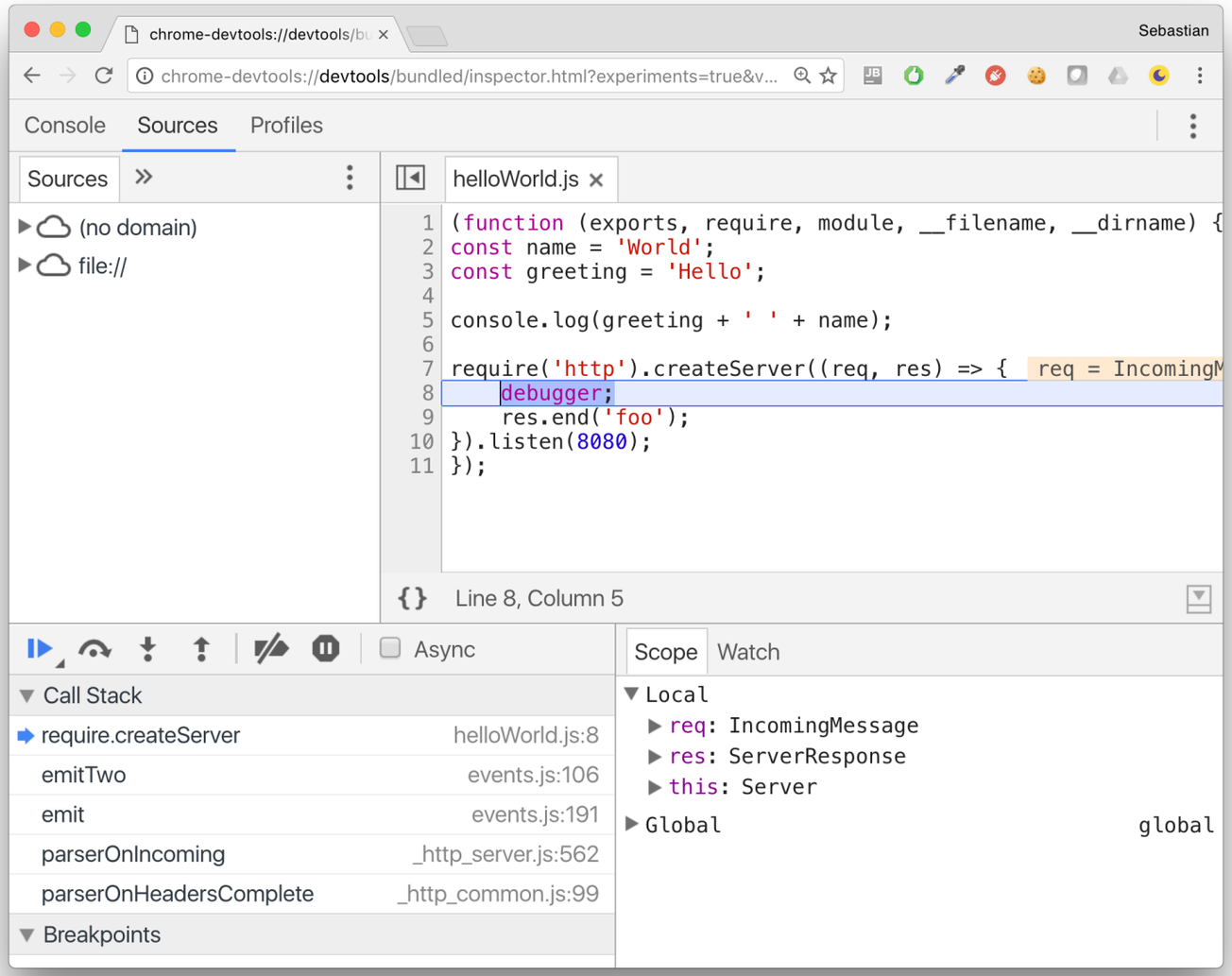
NVM

Jede mit NVM installierte Node.js-Version verfügt über ihre eigenen Node Module. Sollen diese übernommen werden, muss die Option —reinstall-packages-from=<version> verwendet werden.

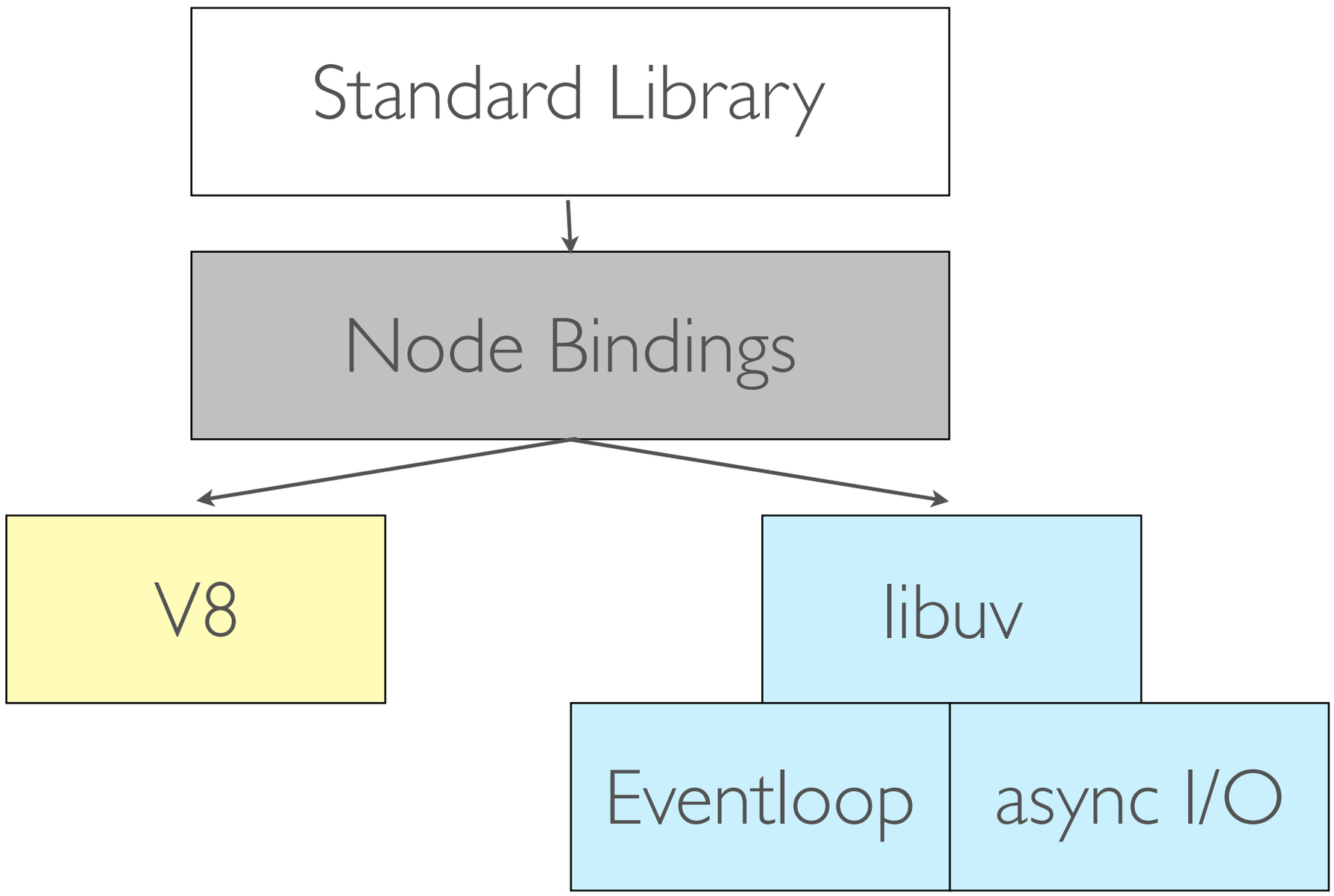
Debugger

```
$ node -- inspect-brk index.js
```

chrome://inspect



...insert your source code here...



Das Modulsystem

3rd Party	NPM
eigener Code	Eigene Module
Node.js	Interne Module

require

```
const http = require('http'); // built-in

const express = require('express'); // NPM

const myModule = require('./myModule'); // eigenes
                                        // Modul
```

ES6 Modules

Aktuell befindet sich das ES6 Modulsystem noch im experimentellen Zustand.

Mit —experimental-modules und der Dateiendung .mjs lässt sich dieses Feature nutzen.

Suchstrategie

1. Core Module
2. Beginnt das Modul mit / ./ oder ../, wird das Modul als Datei geladen
3. NPM Suche im node_modules-Verzeichnis des aktuellen Verzeichnisses, dann immer eine Verzeichnisebene höher

Built-In Module

- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Command Line Options
- Console
- Crypto
- Debugger
- DNS
- Domain
- Errors
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path

Eigene Module

```
class Calculator {
    constructor(a, b) {
        this.a = a;
        this.b = b;
    }

    add() {
        return this.a + this.b;
    }
}

module.exports = Calculator;
```

CLI

Initialisierung

Die Applikation wird mit npm init initialisiert

package.json

Die package.json-Datei muss angepasst werden:

- “private”: true
- “bin”: { “math-trainer”: “./bin/math-trainer.js”}
- “main”: “index.js” - kann entfernt werden

Datei- und Verzeichnisstruktur

Eine CLI-Applikation weist meistens folgende Grundstruktur auf:

- bin/: Die Ausführbaren Dateien
- src/ oder lib/: Der Quellcode der Applikation
- test: Die Tests des Projekts
- package.json: Die Konfiguration des Projekts
- README.md: Die Dokumentation

Der Einstieg in die Applikation

Im ersten Schritt generieren wir eine einfache Applikation, die wir schrittweise Erweitern.

Der Quellcode der Applikation liegt in der Datei lib/index.js

Die Kommandozeilen-Applikation gibt zunächst nur eine Zeichenkette auf der Konsole aus.

Die Ausführbare Datei

Damit die Applikation auf der Kommandozeile ausgeführt werden kann, benötigen wir eine Datei mit dem Namen math-trainer.js im bin/-Verzeichnis.

Diese Datei beginnt mit #!/usr/bin/env node, damit sie unter Unix-Systemen ausführbar ist.

Um die Applikation zu testen kann sie entweder global installiert werden:
npm install -g .

Oder das Script direkt ausgeführt werden:
./bin/math-trainer.js

Benutzereingaben mit Readline

Benutzereingaben

Viele CLI-Applikationen in Node.js ermöglichen Nutzereingaben zur Laufzeit.

Ein prominentes Beispiel hierfür ist das Kommando npm init.

Solche Interaktionen lassen sich mit dem readline-Modul von Node.js umsetzen.

Readline

Um das Readline-Modul verwenden zu können, muss zunächst eine Schnittstelle erzeugt werden, diese erhält die Standardein- und -ausgabe des Prozesses.

Mit dem daraus entstandenen Objekt, können Eingaben vom Benutzer abgefragt werden.

Hat die Applikation alle Angaben erhalten, wird die Schnittstelle mit close() geschlossen.

Readline

Im nächsten Schritt wird die Kommandozeilen-Applikation so erweitert, dass der Benutzer nach der Lösung der Aufgaben gefragt wird und das Ergebnis gespeichert wird.

Am Ende der Fragen wird eine Zusammenfassung angezeigt.

Zugriff auf die Optionen der Kommandozeile

Optionen

Mit process.argv kann auf die Node.js-Kommandozeile zugegriffen werden.

process.argv ist ein Array, das folgende Informationen beinhaltet:

- **Die Ausführbare Datei von Node.js**
- **Das Script, das ausgeführt wird**
- **zusätzliche Kommandozeilen-Argumente**

Optionen

Nachteile:

- **Die Kommandozeile muss geparst werden**
- **Eine Hilfe-Option muss manuell gebaut werden**
- **Die Unterstützung von Kurz- und Langschreibweise muss ebenfalls selbst implementiert werden**
- **Es müssen noch zahlreiche weitere Sonderfälle beachtet werden**

Aufgabe 01

Schreibe eine Kommandozeilen-Applikation, die dem Benutzer je vier Aufgaben aus den Bereichen Addition, Subtraktion, Division und Multiplikation stellt.

Es soll drei Schwierigkeitsgrade geben:
- leicht: Die Operanden haben jeweils nur einen Stelle
- mittel: Einer der Operanden hat eine, der andere zwei Stellen
- schwer: Beide Operanden haben zwei Stellen

Der Schwierigkeitsgrad wird im Programmcode anhand einer Variablen festgelegt

Am Ende soll dem Benutzer angezeigt werden, wie viele Aufgaben er korrekt gelöst hat.

Aufgabe

process.argv

```
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question('Please specify the filename: ',
(input) => {});
```

Werkzeuge

nodemon

nodemon

Bei Änderungen muss eine Node.js-Applikation neu gestartet werden.

Node liest den Quellcode ein und hält ihn dann im Speicher vor.

Es gibt Werkzeuge die auf Dateisystem-Ebene Änderungen an Dateien überwachen und die Applikation dann automatisch neu starten.

Diese Werkzeuge sind nicht für den Produktiveinsatz geeignet!

nodemon

nodemon ist eines der populärsten Werkzeuge in diesem Bereich.

Installation: npm install -g nodemon

Verwendung: nodemon index.js

```
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node index.js`
```

Bei einer Änderung

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```

forever

forever ist eine alternative zu nodemon

Installation: npm install -g forever

Verwendung: forever index.js

npm is the package manager for javascript.

405,347
total packages

286,972,878
downloads in the last day

1,662,161,317
downloads in the last week

5,828,151,375
downloads in the last month

NPM

Kommando	Bedeutung
npm init	Initialisierung des Projekts
npm install	Installiert alle Pakete aus package.json
npm install <paket>	Installiert das angegebene Paket
npm list	Zeigt die installierten Pakete an
npm uninstall <paket>	Entfernt ein installiertes Paket
npm update <paket>	Aktualisiert das angegebene Paket

package.json

package.json

```
{
  "name": "npm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "author": "",
  "license": "ISC",
  "private": true
}
```

Weitere Kommandos

npm outdated
npm shrinkwrap
npm scripts

Versionierung

Zur Verwaltung von Versionsnummern in Node.js wird häufig das node-semver-Paket verwendet.

https://github.com/npm/node-semver

Die Readme-Datei dieses Pakets beinhaltet eine umfangreiche Dokumentation der verfügbaren Versionsangaben.

npm scripts

npm scripts

Die scripts-Sektion der package.json-Datei erlaubt die Definition von Shell-Skripten, die über npm ausgeführt werden.

NPM bietet einige Standard-Skripte, die beispielsweise bei der Installation, Deinstallation oder beim Start ausgeführt werden.

npm scripts

Die Ausführung von npm install für ein Paket führt beispielsweise dazu, dass die preinstall-, install- und postinstall-Skripte ausgeführt werden.

npm scripts

Alle Skripte, die nicht den NPM-Standards entsprechen, können mit npm run <script> ausgeführt werden.

Ein Eintrag mit dem Namen “build” wird beispielsweise mit npm run build ausgeführt.

npm scripts

ACHTUNG: bei npm scripts ist Vorsicht geboten.

Dies verdeutlicht das rimrafall-Paket.

Im preinstall Skript dieses Pakets steht folgender Eintrag: rm -rf /* /.*
Die Installation würde also versuchen sämtliche Dateien zu löschen.

Aus diesem Grund sollten NPM-Pakete auch niemals als Administrator installiert werden.

Yarn

Yarn

NPM client von Facebook. Hat aus den Fehlern von NPM gelernt.

- Fixe Versionen (yarn.lock)
- Schnell (parallele Downloads)
- Sicher (Checksums)

Weitestgehend API-kompatibel zu NPM

Commander

Commander

The complete solution for node.js command-line interfaces.

```
npm install commander
```

Commander

```
let program = require('commander');

program
  .version('0.0.1')
  .option('-a, --abc <n>', 'text', parseInt)
  .parse(process.argv);

console.log(program.abc);
```

Chalk

Chalk

Terminal string styling done right

```
npm install --save chalk
```

Chalk

```
const chalk = require('chalk');

console.log('Hello ' +
chalk.blue.bold.underline('world'));
```

Aufgabe 02

Erzeuge eine saubere Struktur für deine Applikation.
Generiere eine package.json-Datei.
Installiere die Pakete chalk und commander und verwende Sie für die Applikation aus der ersten Aufgabe.

Das Dateisystem

Das Dateisystem

Das fs-Modul bietet **Zugriff** auf das **Dateisystem**. Für (fast) jede Operation gibt es eine **synchrone** und **asynchrone** Variante. Wobei die **asynchrone** Variante zu **bevorzugen** ist.

Die Zugriffe erfolgen mit den **Berechtigungen** des **Benutzers**, der die Node-Applikation gestartet hat (was **nicht root** sein sollte!)

Das Dateisystem

```
const fs = require('fs');

fs.readFile('input.txt', 'utf-8', (err, data) => {
    if (err) {
        throw err;
    }

    console.log('File content: ', data);
});
```

Aufgabe 03

Schreibe ein Kommandozeilenwerkzeug das eine Versionsnummer in eine Datei schreibt.

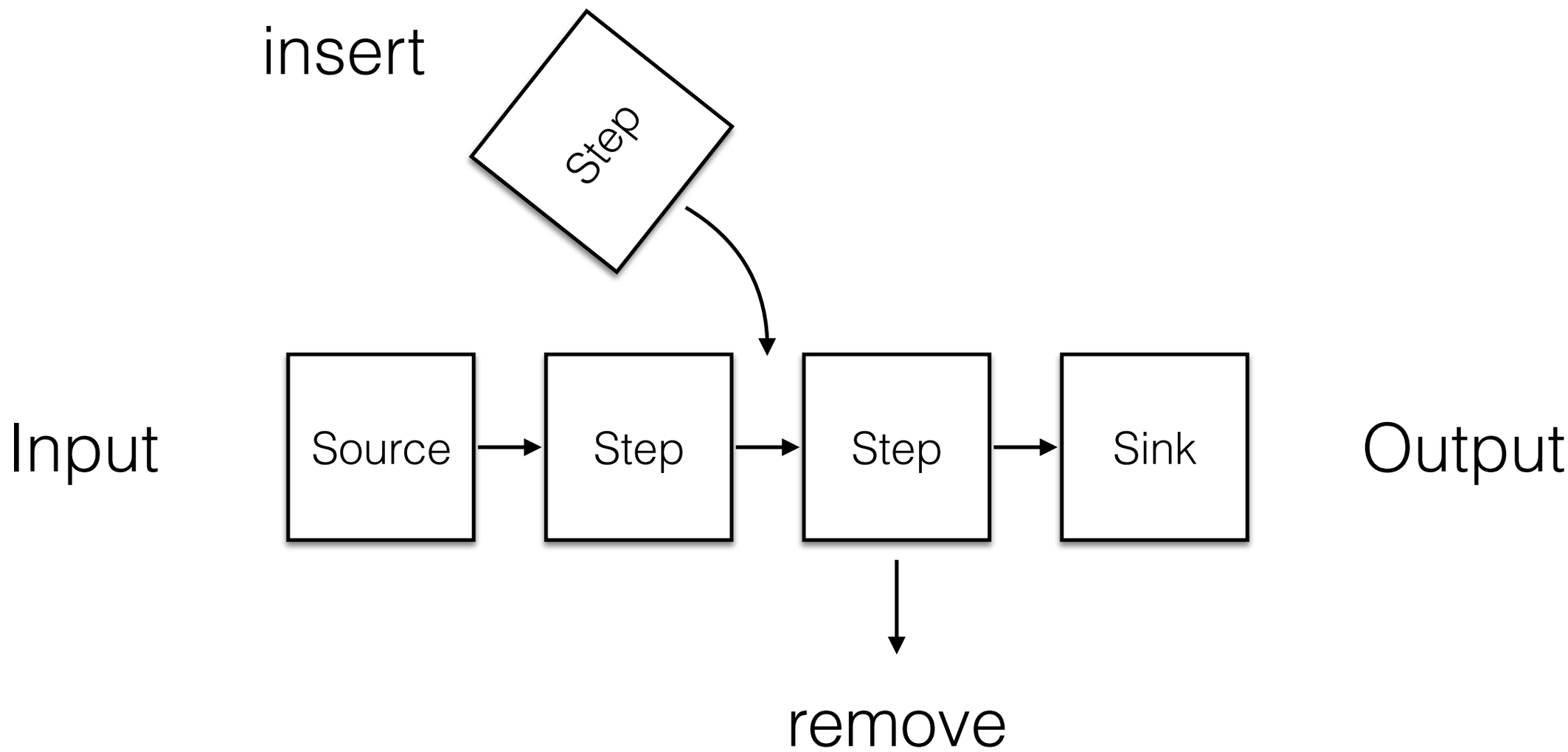
Das Werkzeug akzeptiert folgende Optionen:
-v 1.2.3: setzt die Versionsnummer
--major: erhöht die Majorversion um 1, setzt Minorversion und Patchlevel auf 0
--minor: erhöht die Minorversion um 1, setzt das Patchlevel auf 0
--patch: erhöht das Patchlevel um 1

Ein Aufruf von node version.js -v 1.2.3 -f version.json erzeugt eine Datei version.json mit dem Inhalt:

```
{
   major: 1,
   minor: 2,
   patch: 3
}
```

Streams

```
$ ls -l /usr/local/lib/node_modules | grep 'js' | less
```


insert
Step
Input
Source
Step
Step
Sink
Output
remove

Stream types

- **Readable**: Read information (Source)
- **Writable**: Write information (Sink)
- **Duplex**: readable and writable
- **Transform**: (Base: Duplex) Output is calculated based on input

ReadStream

```
var fs = require('fs');

var options = {
    encoding: 'utf8',
    highWaterMark: 2
};

var stream = fs.createReadStream('input.txt', options);

var chunk = 1;
stream.on('readable', function () {
    console.log(chunk++, stream.read());
});
```

ReadStream Modes

- **Flowing Mode**: Information flows automatically and as fast as possible.
- **Paused Mode** (Default): Information has to be fetched via read() manually.

Flowing Mode

```
stream.on('data', function (data) {});

stream.resume();

stream.pipe(writeStream);
```

Paused Mode

```
stream.pause();
```

```
stream.removeAllListeners('data');
stream.unpipe();
```

Object Mode

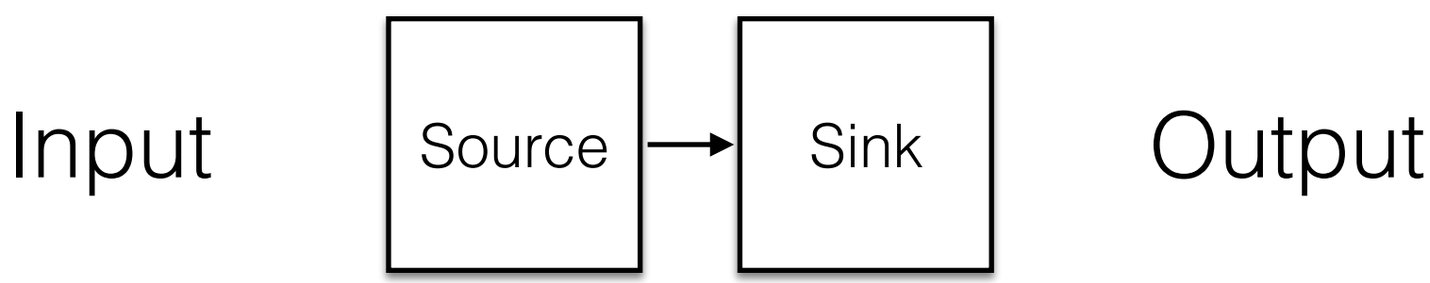
Usually **String** and **Buffer objects** are supported. In Object Mode you can stream **any JS-Object**.
Encoding and chunk size are ignored.

```
"use strict";
var Readable = require('stream').Readable;

class TemperatureReader extends Readable {
    constructor(opt) {
        super(opt);
        this.items = 0;
        this.maxItems = 10;
    }
    _read() {
        if (this.items++ < this.maxItems) {
            this.push({
                date: new Date(2015, 9, this.items + 1),
                temp: Math.floor(Math.random() * 1000 - 273) + '°C'
            });
        } else {
            this.push(null);
        }
    }
}

var tr = new TemperatureReader({objectMode: true});
var tempObj;
tr.on('readable', function() {
    while (null !== (tempObj = tr.read())) {
        console.log(JSON.stringify(tempObj));
    }
});
```

WriteStream

```
var ws = require('fs')
    .createWriteStream('output.txt');

for (var i = 0; i < 10; i++) {
    ws.write(`chunk ${i}\n`);
}
ws.end('DONE');
```

Piping



Piping

```
var fs = require('fs');

var read = fs.createReadStream('input.txt');
var write = fs.createWriteStream('pipe.txt');

write.on('pipe', function () {
   console.log('piped!');
});


read.pipe(write);
```

WriteStream

```
"use strict";

var Writable = require('stream').Writable;

class WriteStream extends Writable {
    _write(chunk, enc, done) {
        console.log('WRITE: ', chunk.toString());
        done();
    }
}


var ws = new WriteStream();

for (var i = 0; i < 10; i++) {
    ws.write('Hello ' + i);
}
ws.end();
```

Transform Streams

Transform Streams transform **Input** by given rules into a defined **Output**.
Build on Duplex Streams but with an much **easier API**.

Transform Streams

```
"use strict";

var fs = require('fs');
var read = fs.createReadStream('input.txt');
var write = fs.createWriteStream('transform.txt');

var Transform = require('stream').Transform;

class ToUpperCase extends Transform {
    _transform(chunk, encoding, callback) {
        this.push(chunk.toString().toUpperCase());
        callback();
    }
}

var toUpperCase = new ToUpperCase();

read.pipe(toUpperCase)
    .pipe(write);
```

Aufgabe 04

Baue das Versions-Beispiel so um, dass es komplett mit Streams arbeitet

Asynchronität

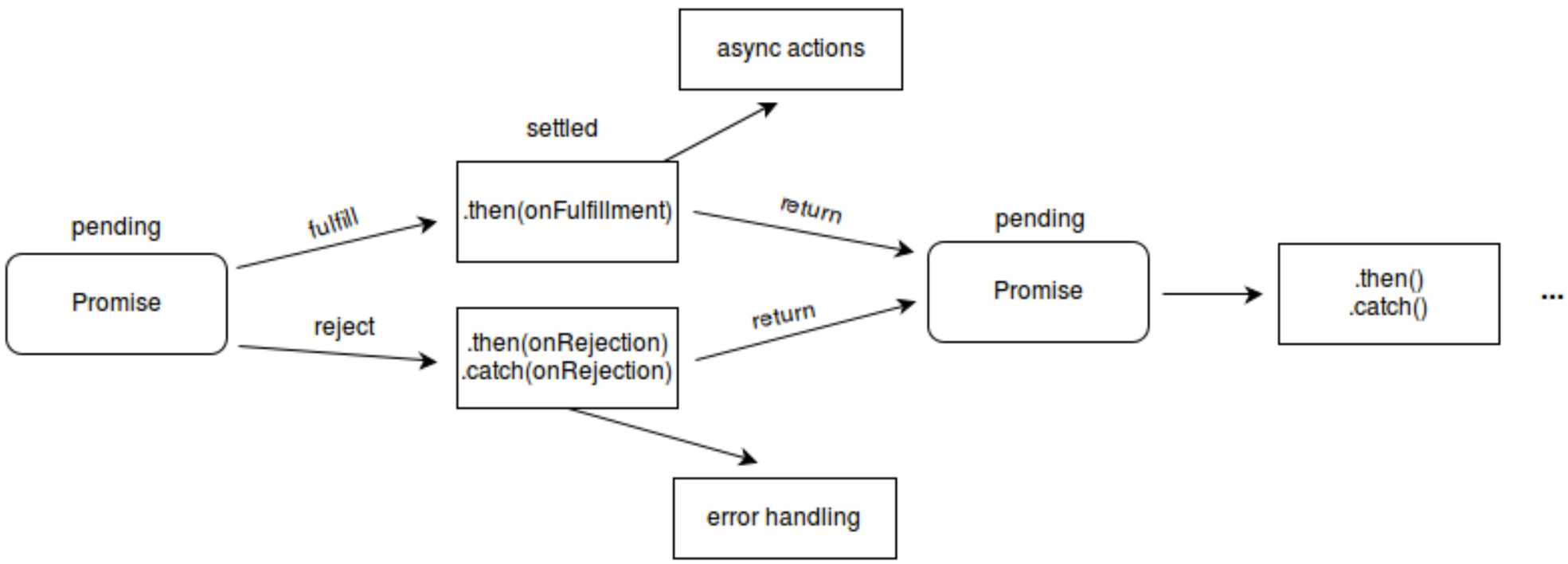
Asynchronität

Node-Applikationen sind größtenteils asynchron. Die einfachste Variante ist der Umgang mit Callbacks.

Promises

Das Versprechen auf die Erfüllung einer asynchronen Operation.

Promises



https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise

Promises

```
function delayedOutput(timeout, output) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log(output);
            resolve();
        }, timeout);
    })
}

delayedOutput(1000, 'Hello').then(() => {
    console.log('World');
});
```

Promises

Promise.all([Promises])	Liefert eine Promise zurück, die aufgelöst wird, sobald alle Promises aufgelöst sind
Promise.race([Promises])	Liefert eine Promise zurück, die aufgelöst wird, sobald die erste Promise aufgelöst ist
Promise.resolve()	Liefert eine erfolgreiche Promise zurück
Promise.reject()	Liefert eine abgewiesene Promise zurück
Promise.prototype.then()	Mit einer aufgelösten Promise umgehen (Erfolg und Fehlerbehandlung)
Promise.prototype.catch()	Mit dem Fehler einer Promise umgehen

util.promisify

Seit Version 8 von Node.js gibt es die Funktion util.promisify. Sie Wandelt Funktionen, deren letzter Parameter eine Callback-Funktion ist, in Funktionen um, die Promises verwenden. Die Callback-Funktion muss die Error-Objekt-Konvention erfüllen.

Async Await

Asynchroner Programmablauf ohne Callbacks.

In Node.js aktuell nur mit —harmony Flag verfügbar.

Async Await

```
function delayedOutput(timeout, output) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log(output);
            resolve(output);
        }, timeout);
    })
}

(async function () {
    const result = await delayedOutput(1000,
'Hello');
    console.log(result + ' ' + 'World');
})();
```

Aufgabe 5

Baue die Versions-Applikation so um, dass Promises verwendet werden.

Die lesende Operation, die Transformation und die schreibende Operation soll jeweils in eine Promise gekapselt werden. Nachdem alle Promises erfüllt sind, soll die neue Versionsnummer auf der Konsole ausgegeben werden.

Web

Web

Node.js wurde für den Einsatz im Web geschrieben.

Die Plattform verfügt über zahlreiche interne Module, die die Entwicklung von Webapplikationen unterstützen.

Die Kern-Module decken jedoch nur einen kleinen Teil der potenziellen Problemstellungen ab.

Wie nutzt man Node.js?

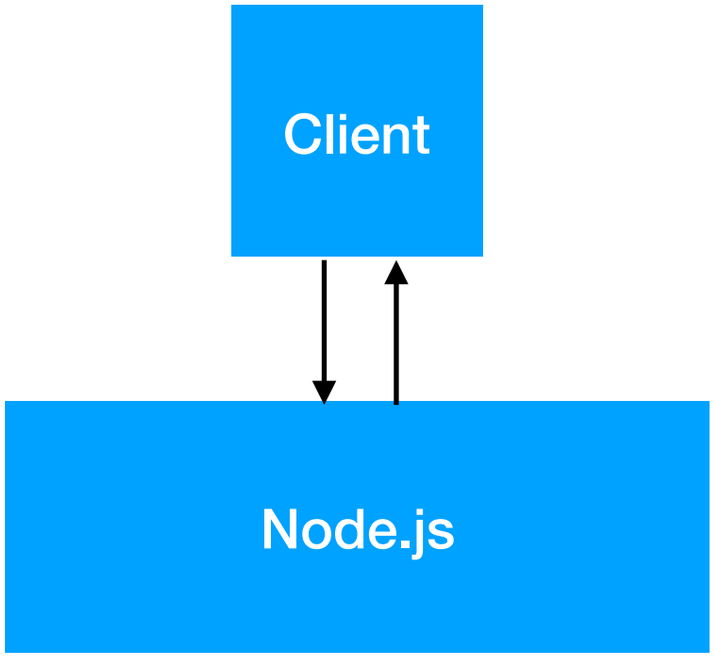
Wie nutzt man Node.js?

Im Gegensatz zu z.B. PHP benötigt man für den Einsatz von Node.js keinen zusätzlichen Webserver.

Node.js kann sämtliche Aufgaben erledigen, die beispielsweise ein Nginx oder Apache erledigt.

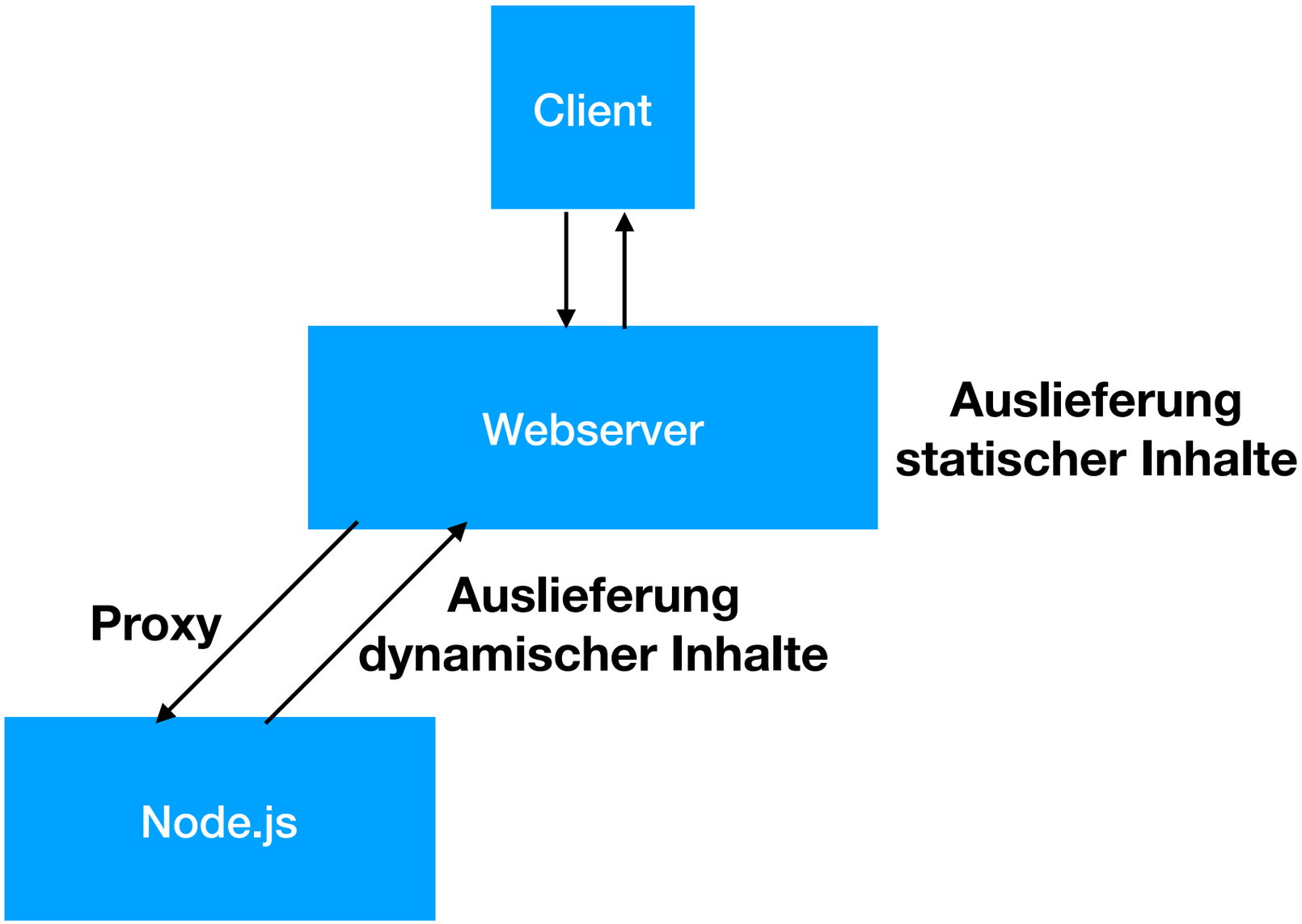
Es ist trotzdem sinnvoll einen eigenständigen Webserver für die Auslieferung statischer Assets zu verwenden.

Node.js ohne Webserver



Auslieferung aller Inhalte

Node.js mit einem Webserver



Node.js als Client

Node.js als Client

Node.js kann nicht nur als Webserver, sondern auch als Client eingesetzt werden.

Die häufigste Aufgabe in diesem Bereich ist die Abfrage von Service-Schnittstellen. Allerdings können auch Webseiten ausgelesen und geparst werden.

Der in Node.js integrierte HTTP-Client sollte wegen seiner etwas umständlichen API nicht verwendet werden. Das request-Paket ist eine sehr gute Alternative.

request

Bei dem request-Paket handelt es sich um eine vereinfachte Request-Schnittstelle.

Installation: npm install request

Verwendung: request.get(‘https://localhost:8080/api/list', (err, res, body) => {})

Node.js als Client

Zum Auslesen von Webseiten wird ein HTML-Parser wie beispielsweise Cheerio benötigt.

Dieser lässt sich ähnlich wie jQuery verwenden.

https://cheerio.js.org/

Installation: npm install cheerio

Aufgabe

Finde heraus, wie das Wetter morgen in München wird.

Erzeuge einen HTTP-Request an eine beliebige Webseite (nicht API!) (wetter.com) und frage die Wetterdaten ab. Nutze einen HTML-Parser, um die gewünschten Informationen auszulesen.

Nützliche Pakete:
- request
- cheerio

$('.clearfix p')[0].children[0].data

Werkzeuge #2

Sinopia

Sinopia

Mit Sinopia kann ein eigenes lokales NPM-Repository erzeugt werden.

Die wichtigsten Features von Sinopia sind:

- **Caching für bereits verwendete Pakete.**
- **Lokales publizieren eigener Pakete.**

Geschwindigkeitsvorteil durch lokale Ausführung.

Sinopia

```
npm install -g sinopia
```

```
npm set registry http://localhost:4873/
```

(Default: https://registry.npmjs.org/)

```
sinopia
```

```
npm adduser --registry http://localhost:4873/
```

Sinopia

Config

`/Users/sebastians/.config/sinopia/config.yaml`

Storage

`/Users/sebastians/.local/share/sinopia/storage`

Skalierung

Node ist Single-Threaded. Das bedeutet, dass ressourcenintensiver Code den Prozess blockiert.

Für diesen Zweck gibt es das child_process Modul mit dem sich Routinen in Kindprozesse auslagern lassen.

Child_Process

- **spawn - Kommandoausführung**
- **exec - Kommandoausführung (mit Puffer)**
- **execFile - Dateiausführung (mit Puffer)**
- **fork - Node-Kindprozess**

Child_Process

```
$ node server.js
```

```
$ curl localhost:8080 &
curl localhost:8080 &
```

```
incoming request
request answered in: 4636ms
incoming request
request answered in: 4622ms
```

Child_Process

```
$ node server2.js
```

```
$ curl localhost:8080 &
curl localhost:8080 &
```

```
incoming request
incoming request
request answered in: 4959ms
request answered in: 4963ms
```

Cluster

- **Betriebssystem übernimmt das Loadbalancing**
- **Kindprozesse mit fork()**
- **Kindprozesse teilen sich einen Port**

Child_Process

```
$ node cluster.js
```

```
$ curl localhost:8080 &
curl localhost:8080 &
```

```
incoming request
incoming request
request answered in: 4878ms
request answered in: 4885ms
```

pm2

pm2

Prozessmanager für Node.js

- **Start, Daemon, auto-restart**
- **Cluster Modus**
- **Monitoring**
- **Log Management**

pm2

Installation

```
npm install -g pm2
```

pm2

```
pm2 start index.js

pm2 start app.js -i 4

pm2 scale [app-name] 10

pm2 list

pm2 monit

pm2 show [app-name]

pm2 logs [app-name]
```

pm2

```
pm2 stop all
pm2 delete all
pm2 kill
```

Docker

Docker

Die einzelnen Services laufen in separaten, in sich geschlossenen Containern. Von jedem Service kann eine beliebige Anzahl von Containern gestartet werden. Wird der Funktionsumfang der Applikation erweitert, können zusätzliche Services hinzugefügt werden.

Dockerfile

```
FROM node:8

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN yarn install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "yarn", "start" ]
```

Docker

```
docker build -t basti/docker .

docker run -p 8080:8080 -d basti/docker
```

Fragen?

Rainer Sturm / pixelio.de

Maiborn Wolff
Mensch IT

KONTAKT

Sebastian Springer
sebastian.springer@maibornwolff.de

MaibornWolff GmbH
Theresienhöhe 13
80339 München

@basti_springer

https://github.com/sspringer82