Workshop
Einstieg in NodeJS

Martina Kraus

23.06.2017

@wing121
github.com/codekittey
kraus.martina.m@googlemail.com

Agenda

8:30 Uhr: Registrierung

9:30 Uhr: Beginn des Workshops
Theorie: Ereignisgesteuerte Architektur, Grundlagen, NPM

11:00 - 11:15 Uhr: Kaffeepause
Streams, Dateisysteme

12:30 - 13:30 Uhr: Mittagessen
Web Applikation mit Node.js und Express

15:00 - 15:15 Uhr: Kaffeepause
Testen mit Node.js / Tooling

Ca. 17:00 Uhr: Ende

Who am I?

```
{
    "name": "Martina Kraus",
    "working": ["Software-Developer at Onwerk GmbH",
                "lecturer at HS Mannheim / HS Karlsruhe"],

    "technologies": ["javascript/ ecmascript", "node", "docker", "nosql-databases" ],

    "others": ["founder of RheinNeckarJS and Docker Mannheim Meetup",
               "Member of Hackerstolz e.V"]
}
```

Wer seid ihr???

Unterlagen

Slides: http://bit.ly/2sHJclp

Alle CodeBeispiele und Übungsaufgaben:

https://github.com/CodeKittey/node_workshop_enterJS

Was ist NodeJS? (In a nutshell)

- Node.js === JavaScript Engine + JavaScript Framework
- v6.11.0 LTS
- V8.1.2 Latest Features
- Entwickelt von Ryan Dahl 2009 (Mittlerweile Joyent Inc.)
- Google V8 Engine (mittlerweile aber auch mit Chakra)
- Ausgelegt für viel I/O
- Event-Driven Architecture
- Asynchron

JavaScript Engines

Engine	Application
V8	Chrome, Node.js
Chakra	Microsoft Edge, Node.js
SpiderMonkey	Firefox
JScript	IE
JavaScriptCore	WebKit, Safari
Rhino	Java Platform

Serverseitiges JavaScript??????

Getting started - Installation

- https://nodejs.org/en/download/
- Windows Installer / Binaries
- Mac OS X Installer / Binaries
- Linux Binaries (.tar .gz)
- Sourcecodes zum selber kompilieren (Abhängig von Python 2.6/2.7)

Getting started - NVM

- Node Version Manager
  - Unterschiedliche Versionen parallel installieren
  - Keine administrativen Rechte

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh
| bash
```

```
Nvm install 6 //installiert die latest node 6 Version
Nvm use 6 // switchen zu einer anderen node Version
```

Getting started - Hello EnterJS

- Über Node.js shell: (**R**ead-**e**val-**p**rint-**l**oop)
  - Nicht geeignet fuer echte Applikationen

- Node.js Anwendung:
  - JavaScript Datei erstellen (index.js)
  - Mithilfe von node aufrufen:

```
$ node index.js
```

Core Module

- Bringt einige Core Module mit:
  - http, net, fs, crypto, buffer

- Einbindung eines Moduls:

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

- Fuer alles andere: **N**ode-**P**ackage-**M**anager

NPM

- Wird mit der Installation mitgeliefert
- Abhängige Module können in einem JSON-File definiert werden: `package.json`

- Dieses wird mit `npm init` initialisiert

Package.json

```
{
 "name": "node-js-sample",
 "version": "1.1.",
 "description": "A sample Node.js app using Express 4",
 "main": "index.js",
 "scripts": {
   "start": "node index.js"
 },
 "dependencies": {
   "express": "^4.13.3"
 },
 "devDependencies": {
   "mocha": "^4.13.3"
 },
  "license": "MIT"
}
```

NPM

- Installieren von Npm-Modulen (und deren Abhängigkeiten) :

```
$ npm i(nstall) <modulName>
$ npm i(nstall) --save <modulName> (Eintrag in Package.json)
$ npm i(nstall) --save-dev <modulName> (Eintrag in Package.json)
```

- Installieren von globalen Modulen

```
$ npm i(nstall) -g <modulName>
```

node_modules

- Dort werden alle via npm installierten Packete abgelegt.

- Jedes Projekt hat seinen eigenen node_modules Ordner

1. Node Core Module
2. npm-Module
3. Selbst Geschriebenen Module (lokale Dateien)

Module

- Anwendungen in logische Teile Blöcke (Module) aufspalten)
- Einzelne Module können wiederverwendet werden
- Leichter zu testen (Unit-Tests)
- Strukturierung des eigenen Sourcecodes

Module in Node.js

- Inkludieren von eigenen Modulen:

```
const myModule = require(‘<path-to-module>/myModule’);
```

- Module Verfuegbar machen:

```
module.exports = myModule;
```

Module in Node.js

```
// doo.js
var Doo = function () {};

Doo.prototype.log = function () {
    console.log('doo!');
}

module.exports = Doo;
```

```
// app.js
var Doo = require('./doo.js');
var doo = new Doo();
doo.log();
```

Do it yourself!!!

- Schreibe ein Taschenrechner-Modul (plus / minus / geteilt / multipliziert)
- Dieses Modul soll in eine app.js importiert werden und wie folgt genutzt werden:

```
console.log(calculator.add(number_one, number_two));
console.log(calculator.divide(number_one, number_two));
console.log(calculator.multiply(number_one, number_two));
console.log(calculator.substract(number_one, number_two));
```

Node.js Architektur

- Ausgelegt für viel I/O
  - Prinzip der Nicht blockierende I/O
  - Asynchron (Callbacks)

- Ereignisgesteuerte Architektur

Traditionelles I/O (synchron)

```
var db_result = db.query(“select * from table“);
doSomething(db_result); //wait!
doSomeOtherVoodoo();
//blocked
```

Non-blocking I/O (asynchron)

```
db.query(“select * from table“,function(db_result) {
    doSomething(db_result); //wait
});

doSomeOtherVoodoo(); //don’t have to wait
```

Callback Konzept (in a nutshell)

- Funktionen sind Objekte,
- Sie können als Parameter einer anderen Funktion übergeben werden.
- Funktion zur Auswertung wird direkt mitgegeben,
- Verhindern das blockieren von Programmen

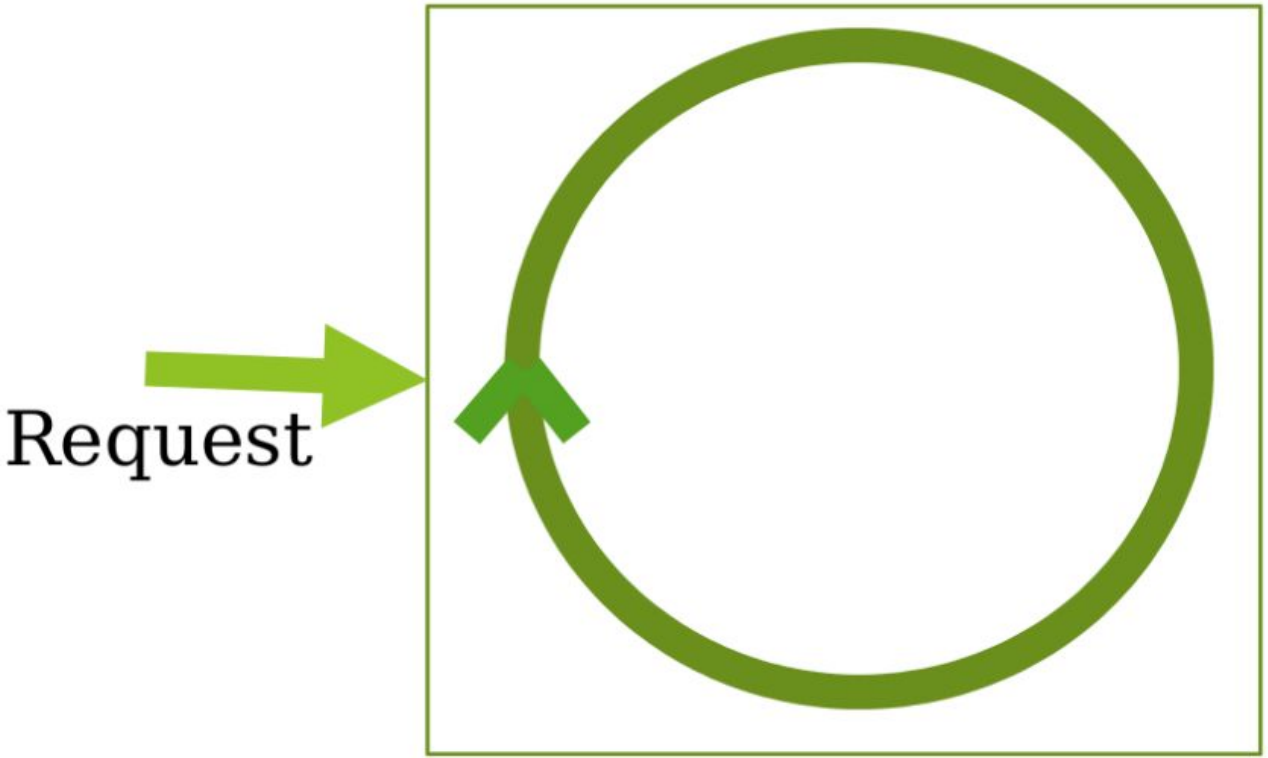
Verarbeitung von Requests

- Prozessmodell (Apache 1.x)
- Threadmodell (Apache 2.x)
- **Event-Driven Architecture (NodeJS und Nginx)**

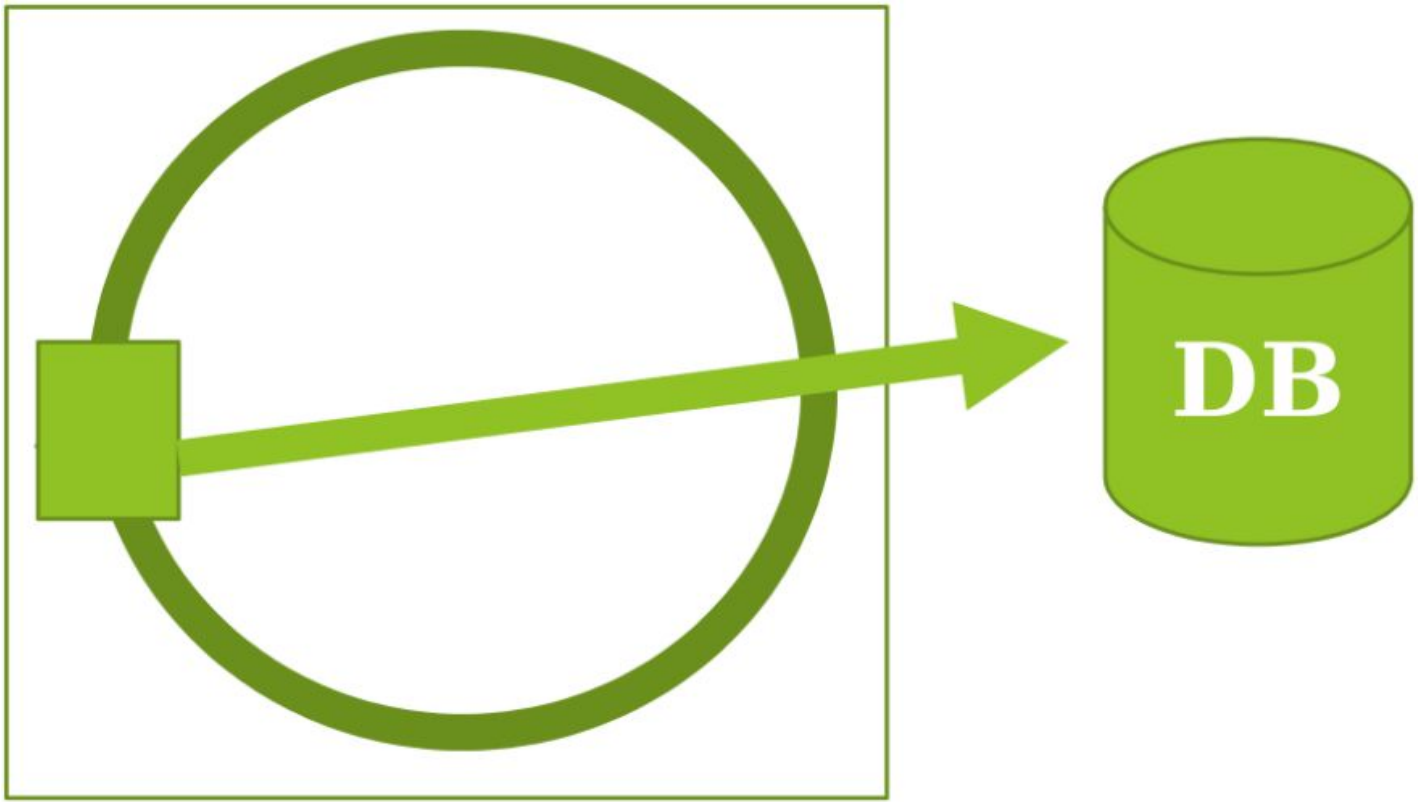
Event-Loop

- Selbstgeschriebene Programme in Node.js laufen “Single-Threaded”
- Node.js läuft asynchron
  - Mit setTimout() kann allerdings ebenso blockiert werden
- Node.js Applikationen laufe in einer Event-Loop
  - Wartet auf Events
  - Triggert eine Callback Funktion

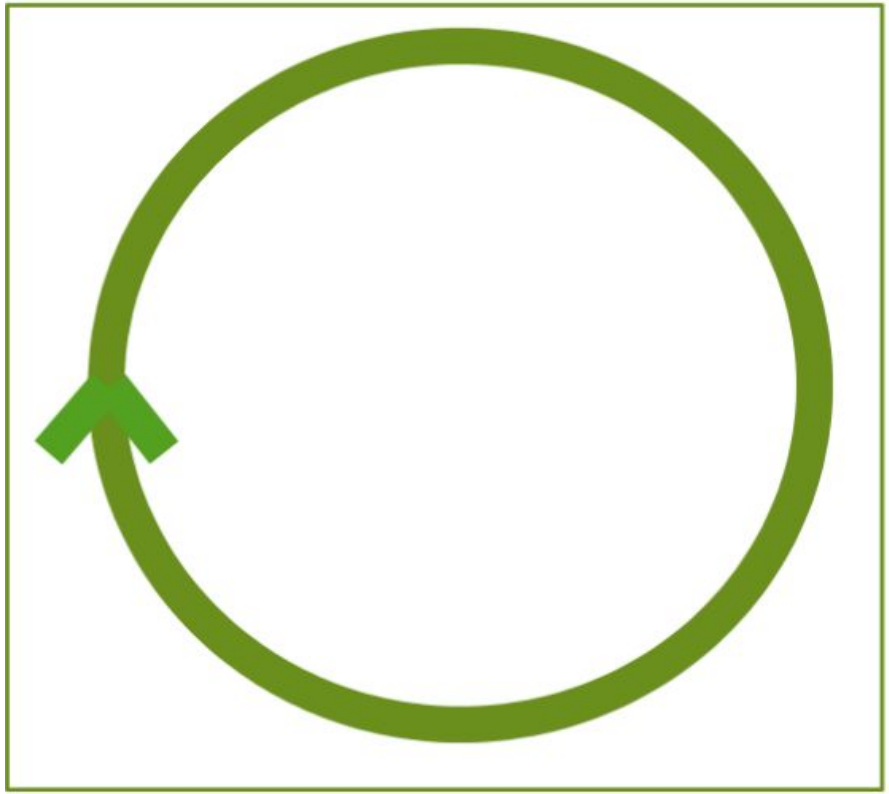
Event-Loop



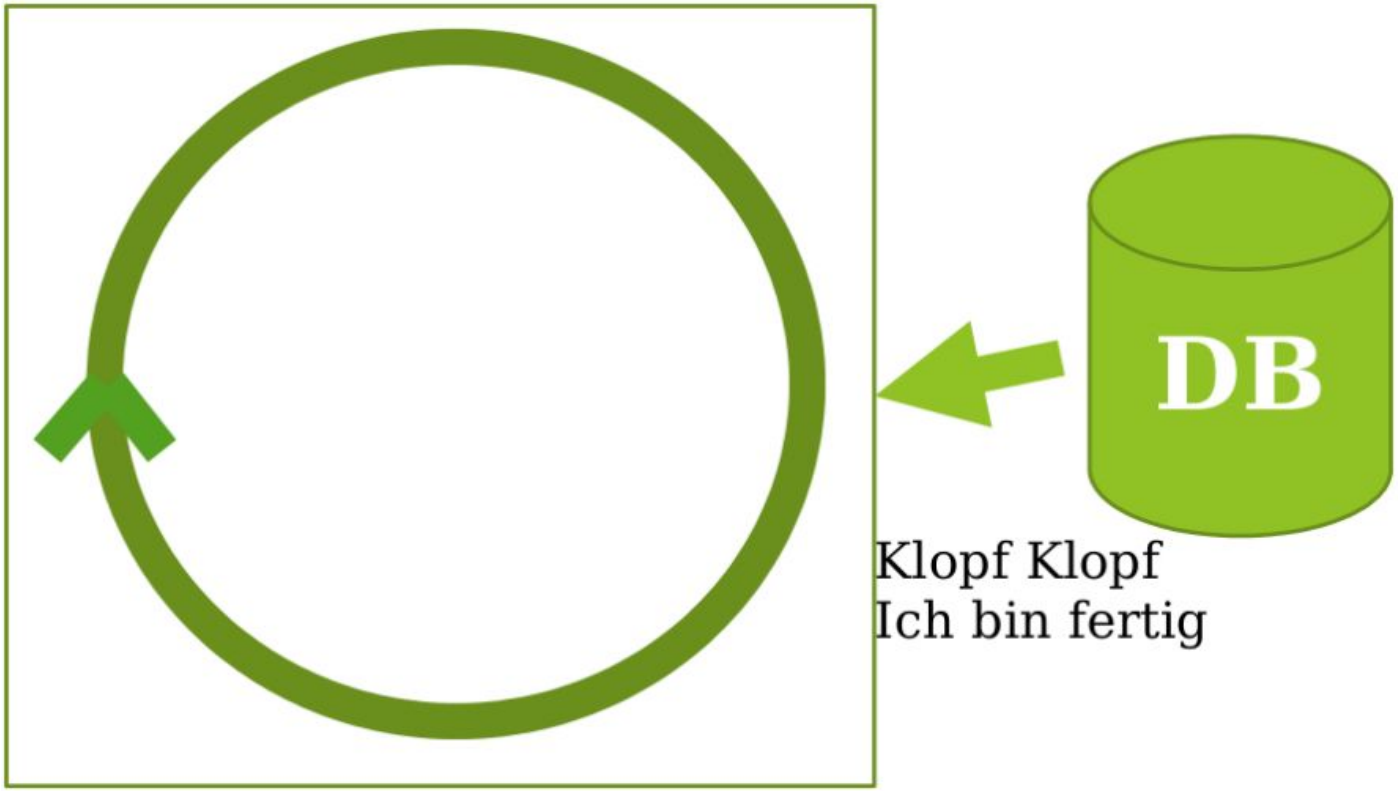
Event-Loop



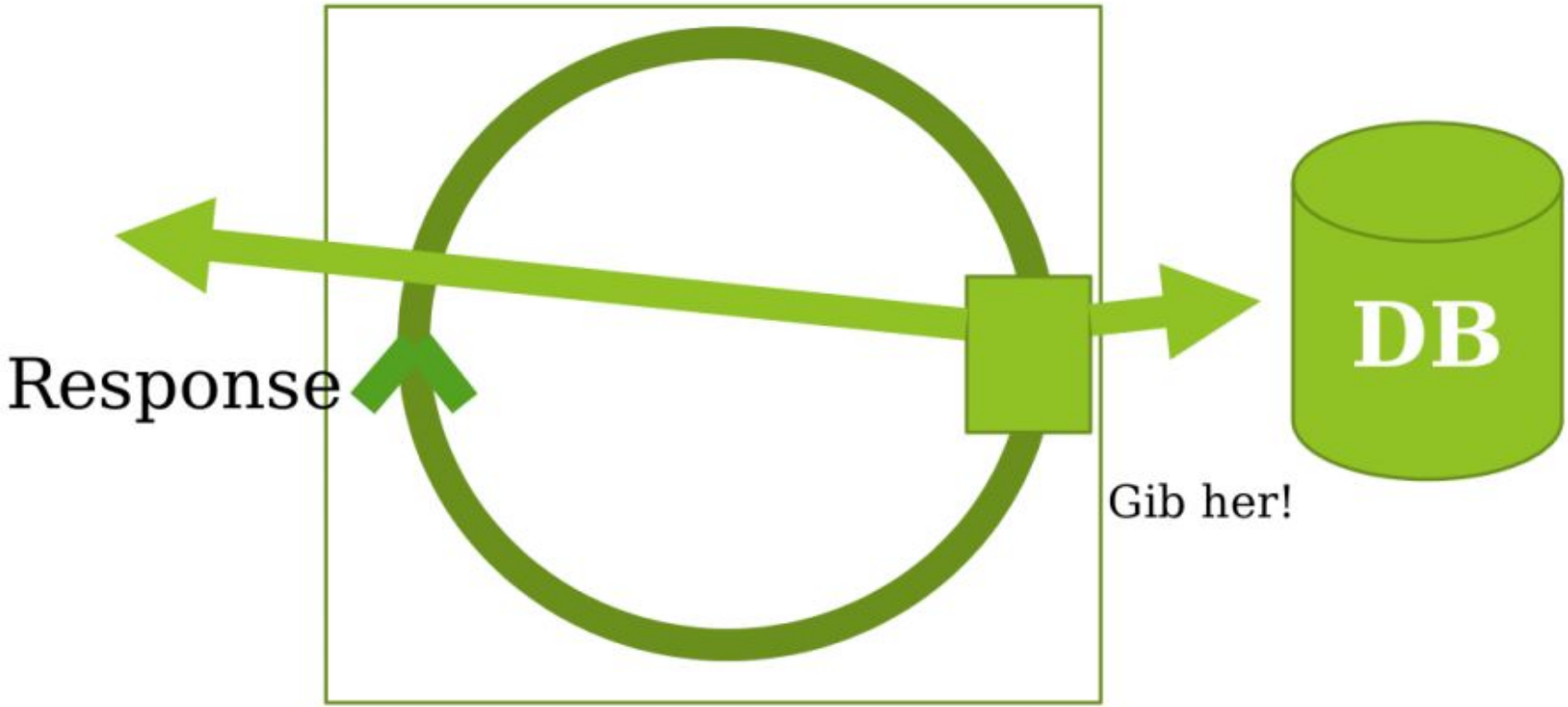
Event-Loop



Event-Loop



Event-Loop



EventEmitter

- Node.js Core Module
- lauschen auf Events:

```
eventEmitter.on('data_received', function(){
   console.log('data received succesfully.');
});
```

- feuern von Events

```
eventEmitter.emit('data_received');
```

Teil 2:
Dateisysteme/ Streams

File System

- Node Core Modul fs

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

- Ermöglicht den Zugriff auf Dateien und Ordner
- Verzeichnisse Lesen (synchron und asynchron)
  `fs.readdirSync || fs readdir`
- Dateien Lesen (synchron und asynchron)
  `fs.readFileSync || fs readFile`

path-Modul

- Node Core Modul fs

```
const path = require(‘path’);
```

`path.join()`: fügt Pfade plattformunabhängig zusammen

Globale Objects

- `__dirname`
  - Verzeichnis der aktuellen Datei
- `__filename`
  - __dirname + Name der aktuellen Datei
- `process.cwd`
  - Aktuelle Arbeitsverzeichnis

Dateien schreiben

```
fs.writeFile();

fs.writeFileSync();
```

- Falls Error während dem Schreiben:
  - Kein “all or nothing”
  - Halb-fertige Datei
  - Abhilfe durch voriges umbenennen

Informationen über Datei auslesen

```
fs.stat();
```

- Liefert Meta-Informationen über eine Datei:
  - Größe
  - Aktualisierungs-Datum
  - Erstellungsdatum

Do it yourself!!!

- Erstelle eine Datei *test.txt* mit folgendem Inhalt:
  “EnterJS rockz”
- Schreibe die Meta-Informationen des Files test.txt in eine neue Datei mit dem Namen “meta_test.txt”

Streams

- Kontinuierlicher Fluss an Daten
- Größe und Ende ist im Voraus nicht bekannt
- Streams ähneln Arrays
- Haben eine zeitliche Reihenfolge
- Verarbeitung von größeren Dateien moeglich

Streams

- Jeder Stream ist eine Instanz des EventEmitters
- Node bietet vier unterschiedliche Streams an:
  - **Readable:** für Lese-Operationen
  - **Writable:** für Schreiboperationen
  - **Duplex**: für Lese- und Schreiboperationen
  - **Transform:** duplex-stream, bei dem der Output abhängig von dem Input ist

Readable-Streams

- Events:
  - **data**: Sobald Daten gelesen werden können
  - **end**: Sobald der Datenstrom beendet ist

```
readableStream.on(‘data’, callback)
readableStream.on(end, callback)
```

Writeable-Streams

- Funktionen:
  - **write**: schreiben in den Stream
  - **end**: beenden des Streams

```
writeableStream.write(‘data’) //return true oder false
writeableStream.end()
```

Writeable-Streams (Synchronisierung)

- Problem-Szenario:
  - Readable Stream übergibt Daten an den Writable Stream (Lesen ist schneller als schreiben)
  - Schreibpuffer läuft über -> Datenverlust
  - Readable Stream mit *pause()* anhalten.
  - Event: *drain,* sobald Writable Stream wieder Daten aufnehmen kann
  - Readable Stream mit *resume()* fortsetzen

Pipe

- Synchronisiert Streams automatisch
- Verkettet Streams

```
readerStream.pipe(writerStream);
```

Duplex-Stream

- Ein “Behälter” für writeable und readable Streams
- Beide Streams in einem Objekt
- Eher selten
- Streams sind unabhängig voneinander

Transform-Stream

- Ein “Behälter” für writeable und readable Streams

writableStream -> Transformation / Manipulation ->

readableStream,

Do it yourself!!!

- Implementiere folgendes Szenario mit Streams:
  - Eine Datei input.txt wird eingelesen
  - Die Daten der eingelesenen Datei sollen in eine neue Datei output.txt geschrieben werden
  - Beachte hierbei die Synchronisation der Daten
    - drain-event
    - `const isReady = writerStream.write(data,'UTF8');`

Teil 3:
Web APIs in Node.js/ Express

Web APIs - Früher

- Serverseitiges rendering, statische Webseiten
- Das gesamte HTML wird immer mit übertragen
- Viele Daten -> viel Netzwerktransfer

Page	**Size**	**EDGE** **max. 384 kBit/s**	**UMTS** **max. 2 MBit/s**	**LTE** **max. 100 MBit/s**
Google.de	500 kB	10 seconds	2 seconds	~0 seconds
Twitter	2,5 MB	54 seconds	10 seconds	~0 seconds

- Keine Offline Fähigkeit

Web APIs - Früher

```
var html = new Array();
html.push('<div class="band-info"><h2>');
html.push(band.bandName + '</h2>');
$.each(band.members,function() {
  html.push('<span class="band-member">');
  html.push(this.name + ' - ');
  html.push(this.instrument + '</span>');
});
if (band.single) {
  html.push('<a href="' + band.single.url);
  html.push('">Download "');
  html.push(band.single.title + '"</a>');
}
html.push('</div>');
document.append(html.join(""));
```

Web APIs - Heute

- Reine Datenübertragung (meist JSON)
- HTML wird clientseitig gerendert
- Daten werden via JavaScript in das HTML gemerged
- Entlastung des WebServers

Web APIs - Heute

- REST API, HTTP API, *** API
- Kleine Modulare Service-Schnittstellen
- Services können sich gegenseitig aufrufen
- Sicherheit über Authentifizierungs-Token
- Jeder Client der HTTP spricht kann mit dem Service interagieren

Web APIs - Basics

- HTTP Verben
  - **GET**: Lädt Daten von einem WebServer
  - **PUT**: Aktualisiert Daten auf einem WebServer
  - **POST**: Legt neue Daten auf einem WebServer an
  - **DELETE**: Löscht Daten auf einem WebServer
- HTTP Web server
- Uniform Resource Locator URL

Web APIs - Schnittstelle

Beispiel:

https://localhost/api/v1/blog/11

Eindeutiger Bezeichner der Datei/ Ressource

GET lädt Blogartikel mit der Id 11

PUT aktualisiert den Blogartikel mit der Id 11

DELETE löscht den Blogartikel mit der Id 11

POST legt einen Blogartikel an (in dem Fall würde keine Id benötigt werden)

Web APIs - Schnittstelle

Mehr Beispiele:

- **GET** /api/blog - gets all blog articles
- **DELETE** /api/user - deletes all users

Filterung via Queryparameter:

- **GET** /api/blog?search=cat - get all blog articles with its content “cat”
- **GET** /api/blog?orderBy=date&direction=desc - get all blog articles ordered by date descending
- **GET** /api/blog?page=3 - gets the third page

Web APIs - Anforderung an WebServer

- APIs zur Verfügung stellen.
- Routingschema der URL != Ordnerstruktur des WebServers
  - Mapping der Routen auf die Ressourcen
  - Client gibt via MIME Type an wie er die Daten haben möchte (JSON, XML)
  - Server muss die Ressourcen dementsprechend umwandeln.
- Unterstützung aller HTTP Methoden

HTTP-Modul

- Node.js Core Modul
- API zur Implementierung eines WebServers

```
const server = http.createServer(requestHandler);

server.listen(1337, ‘127.0.0.1’);
```

HTTP-Modul

```
function requestHandler(request, response) {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
}
```

Express

- Serverseitiges Web Application Framework für Node.js
- Entwickelt von Douglas Christopher Wilson and community
- Aktuelle Version: 4.0 (MIT License)
- Erweitert Node.js an Funktionalität
  - Ausliefern statischer Dateien (CSS/ JS/ HTML)
  - Routing
  - Aufbau des Response

Simple WebServer mit Express

```
const express = require('express');
const app = express();

app.get('/', function(req, res) {
       res.end('Hello World!')
   });

app.listen(1337);
```

Simple WebServer mit Express

Laden des Moduls Express in der JavaScript-Server Datei

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

Requesthandler wenn ein Request mit der HTTP-Methode GET für die URL localhost/home eingetroffen ist

```
app.get('/home', function(req, res) {
    res.end('Hello World!')
})
```

Basic Routing mit Express

```
app.METHOD(PATH, HANDLER)
```

METHOD: eine Http Methode

PATH: aufrufender Pfad

HANDLER: Requesthandler

```
app.get('/', function (req, res) {
res.send('Hello World!');
});
```

Basic Routing mit Express

POST

- Legt Dateien an
- Daten über Request-Body
- Nicht sicher, nicht idempotent

```
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```

Basic Routing mit Express

PUT

- Legt Dateien an
- Daten über Request-Body
- Nicht sicher, aber idempotent

```
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
});
```

Basic Routing mit Express

DELETE

- Löscht eine Datei
- Nicht sicher, aber idempotent

```
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
});
```

Basic Routing mit Express

Regular Expressions sind als Pfad gültig

```
app.get('/ab*cd', function (req, res) {
res.send('Hello World!');
}); → match abcd, abxxd, abRABDOMcd, ab123cd usw.
app.get('/ab(cd)?e', function (req, res) {
res.send('Hello World!');
}); → match  /abe und /abcde.
```

Basic Routing mit Express

Variablen

Jede Ressource ist eineindeutig über ihre URL identifizierbar

```
app.get('/users/:id?', function(req, res){
    var id = req.params.id;
    if (id) {
        // do something
    }
});
```

Basic Routing mit Express

Variablen

Jede Ressource ist eineindeutig über ihre URL identifizierbar

```
app.get('/users/:id?', function(req, res){
    var id = req.params.id;
    if (id) {
        // do something
    }
});
```

Request Objekt API

Attribut	Beschreibung
req.baseUrl	URL zur Ressource
req.body	Request Body
req.cookies	Client Cookies
req.hostname	Host name des HTTP Headers
req.params	Alle Url-Parameter
req.get(<http-header-key>)	req.get(‘content-type’)

Response Objekt API

Attribut	Beschreibung
res.end	Beendet einen Response ohne Body
res.status	Setzen des Http-Statuscode
res.json	Sendet einen json-Body
res.send([body])	Sendet Reponse mit einem Body
res.set(<http-resoonse-header>)	res.set('Content-Type', 'text/plain');
res.redirect	Redirect zu angegebenem Pfad

Express Middleware

- Bearbeitet den Request bevor an die Requesthandler gegeben wird
- Können selbst geschrieben werden
  - Express bietet allerdings einige an

```
app.use("/login", webguiMiddleware);
app.use("/login", versionMiddleware);
app.use("/weather", versionMiddleware);
```

Ausliefern statischer Dateien

Konfiguration der Express-App

```
app.use(express.static(path.join(__dirname, "public")));
```

Mit app.use können Express-Parameter zur Konfiguration übergeben werden
path: NodeJS Core Modul für Handling von Applikationspfäden.

```
const path = require("path");
path.join([path1][, path2][, ...]) //Konkatenation der strings
```

express.static(<dirname>);

Gibt an unter welchem Dateipfad die statischen
Dateien zu finde sind. (hier: /applicationRootPath/public/)

Middleware: Body-Parser

- Wichtig bei PUT und POST-Methoden
- Ist in req.body gespeichert
- Der Request Body beinhaltet nur Key-Value Paare
  - Schlecht auszulesen (kein Objekt)
  - Kein encoding
  - Keine “Konfigurationsmöglichkeiten” des Bodys

Middleware: Body-Parser

Installation

```
$ npm install body-parser
```

API

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

Express Konfiguration zur Nutzung:

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


app.post('/profile', function (req, res) {
    console.log(req.body);

    res.json(req.body);

    });
```

Weitere Express middleware

cookie-parser	Formatierung des Request-Cookies
compression	Komprimierung des HTTP Response
cors	Aktiviert cross-origin resource sharing
serve-favicon	Liefert das Favicon aus
session	Stellt eine Session her

Do it yourself!!! - WebAPI

- https://github.com/CodeKittey/node_workshop_enterJS
- ExpressExercise

1. Schreibe einen WebServer der für die Route ‘/’
   die Datei index.html ausgibt.

2. Schreibe ein WebAPI mit Express mit folgenden Routen:

Do it yourself!!! - WebAPI

`GET /api/v1/players`
Liefert alle auf dem Server gespeicherten Spieler

`GET /api/players?favorites=true`
Liefert alle auf dem Server gespeicherten Spieler,
welche als Favorit markiert wurden

`GET /api/players?search=<char>`
Liefert alle auf dem Server gespeicherten Spieler die mit dem Anfangsbuchstaben (Nachnamen) beginnen welcher mit hilfe des search Flags in der URL mitgegeben wird
(kann hierbei Werte von A-Z annehmen)

Teil 4:
Node.js Tools / Testing

Node.js Development Tools

- Eslint (statische Code Analyse)
- Mocha/ Jasmine/ Karma/ Protactor (Unit Tests)
- Grunt/ Gulp/ Webpack (Bundling)

Eslint

- Tooling zur statischen Code Analyse

Installation:

```
$npm install eslint --save-dev
$npm install -g eslint
```

- Regeln zur Analyse werden in einer .eslintrc Datei festgelegt

```
$eslint --init
//.eslintrc Datei wird erstellt (mit Default Rules)
```

Eslint - Regeln

- Vollständiges Regelset:
  - http://eslint.org/docs/rules/

```
$eslint app.js
//app.js wird anhand definierter Regeln geprüft

$eslint --fix app.js
//app.js wird anhand definierter Regeln geprüft und teilweise
//automatisch korrigiert
```

Do it Yourself!!!

- Unter /tools/ ist eine .eslintrc Datei zu finden.
- Überprüfe deine JavaScript-Files für WebAPI mithilfe von Eslint

Eslint für IDEs

- Atom:
  - https://atom.io/packages/linter-eslint
- WebStorm:
  - https://plugins.jetbrains.com/plugin/7494-eslint
- Sublime:
  - https://github.com/roadhump/SublimeLinter-eslint
- Visual Studio Code
  - https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint

.eslintignore

- Auflistung von Ordnern und Dateien die nicht mit geprüft werden sollen
- Aehnlich dem .gitignore Konzept
- node_modules werden hierbei automatisch ignoriert

Testen mit Node.js

- **Test runner:** mocha, tape, protector
- **Assertion Library:** chai, assert
- **Test Setup (Erstellung Mock-Objekte):** sinon

Mocha

simple, flexible, fun

- Umfassendes Test framework
- Ermöglicht asynchrones Testen
- Modultests
- Testen von WebAPIs

```
$ npm install -g mocha
$ npm install --save-dev mocha
```

Mocha - Assertion

simple, flexible, fun

- Vergleiche mithilfe assertion libraries:
  - Should.js
  - Expect.js
  - Assert.js
- Assertion Library Chai kombiniert alle drei Libraries

Chai

simple, flexible, fun

- BDD / TDD Assertion Library für node.js

$ npm install --save-dev chai

```
doo.should.be.a('string');
doo.should.equal('bar');
doo.should.have.lengthOf(11);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

Testen von WebAPIs

simple, flexible, fun

- chai-http

$ npm install --save-dev chai-http

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const should = chai.should();
chai.use(chaiHttp);
```

Testen von WebAPIs

simple, flexible, fun

- Testen eines Get-Requests:

```
chai.request(server)
        .get('/api/v1/players')
        .end((err, res) => {
            res.should.have.status(200);
            res.body.should.be.a('array');
            res.body.length.should.not.be.eql(0);
            done();
        });
```

Testen von WebAPIs

simple, flexible, fun

- Testen eines Post-Requests:

```
chai.request(server).post('/players')
                    .send(employee)
                    .end((err, res) => {
                        res.should.have.status(201);
                        res.body.should.be.a('object');
                        res.body.id.should.be.a('string');
                        done();
                    });
```

Hooks

simple, flexible, fun

```
describe('hooks', function() {
  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach(function() {
    // runs before each test in this block
  });

  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

Do it yourself

simple, flexible, fun

- Unter TDD_Exercise befindet sich eine *server.js*-Datei und eine passende Test-Datei.

1. Schreibe zuerst die Tests zu den jeweiligen Routen
2. Implementiere den WebServer

@wing121

github.com/codekittey

Nuetzliche Links

- http://nodejs.org/download/ (Node.js Homepage)
- https://www.joyent.com/noderoad (next Steps Node.js)
- https://github.com/creationix/nvm/blob/master/README.md
- http://nodeschool.io/ (Tutorials)
- https://www.npmjs.org (Node packaged Modul – Packed Manager)