

Multimedia Engineering II

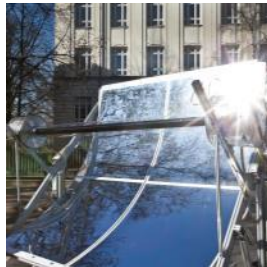
04 REST APIs mit node.js

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Agenda

- **Wiederholung**
- **Update zum Semesterplan**
- **Nachtrag: REST Schnittstellen generieren, testen, dokumentieren**

- **Ziel: TwitterApp REST-API**
- **Aufbau einer nodejs Anwendung**
 - **Requires**
 - **Middleware: Unterschied .use(), .get()**
 - **Request und Response**
 - **Parameter auslesen**
 - **Fehlerbehandlung**
- **Code-Übung zum Aufbau der API**

- **Kapselung von Routes mit express.Router()**
- **Postman im Einsatz**
- **Modul nodemon**

- **Zusammenfassende Fragen**
- **Ausblick**

Wiederholung API – Was sind die Merkmale guter APIs?



- Gute Dokumentation (idealerweise selbsterklärend)
 - Sonst praktisch unbrauchbar
- Langfristig
 - Stabile Schnittstelle auf die „gebaut“ werden kann
- Unabhängig
 - Kann von verschiedenen Endsystemen genutzt werden
- Erweiterbar
 - Versionierung für zukünftige Veränderungen

= **GLUE** (engl. für Kleber, ..der die Sachen zusammenhält)

Wiederholung

- **Aus welchen drei wesentlichen Bausteinen besteht eine REST-Schnittstelle?**



Wiederholung

- **Welcher Zusammenhang besteht zwischen HTTP Methoden und RESTful APIs?**

CRUD	HTTP Methode
Create	POST
Read	GET
Update	PUT
Delete	DELETE

+ PATCH

Wiederholung

- Was ist HATEOAS und was hat es zu tun mit REST?
- **Hypermedia as the engine of application state**
- **Sendet in Antwort**
 - neben den Ressourcen
 - **AUCH Verweise** (auf weitere Ressourcen und Operationen)
- Komplettes HATEOAS ist Overkill
- aber in Teilen nützlich (siehe REST Design Richtlinien 5 u. 6)

Wiederholung

■ Welche Design Guidelines für RESTful Design kennen Sie?

1. Rückgabetyp
2. API Versionierung
3. HTTP Status und Fehler
4. Sicherheit
5. Filtern und Blättern
6. Verweise und Expansion

Agenda

- Wiederholung
- Update zum Semesterplan
- Nachtrag: REST Schnittstellen generieren, testen, dokumentieren

- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
 - Requires
 - Middleware: Unterschied `.use()`, `.get()`
 - Request und Response
 - Parameter auslesen
 - Fehlerbehandlung
- Code-Übung zum Aufbau der API

- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon

- Zusammenfassende Fragen
- Ausblick

■ Zeitplan Zug 1, Dienstag (vorläufig)

	Datum	Thema
1	05.04.2016	
2	12.04.2016	Einführung, Ziele, Ablauf, Benotung
3	19.04.2016	Client-Server Architektur
4	26.04.2016	REST-APIs
5	03.05.2016	REST in node.js
6	10.05.2016	Debugging und Testen
7	17.05.2016	Strukturierung, Modularisierung
8	24.05.2016	Vertiefung einzelner Themen
9	31.05.2016	Datenhaltung, SQL, NoSql, primär mit MongoDB
10	07.06.2016	backbone.js als Gegenpart zu REST/node
11	14.06.2016	Authentifizierung und Patterns
12	21.06.2016	Mobile Development/Cross-Plattform-Development
13	28.06.2016	Gastdozent(en)
14	05.07.2016	Klausurvorbereitung
15	12.07.2016	Klausur PZR1 (Di, 12.07. 14:00 Uhr , Ingeborg-Meising-S.)
16	19.07.2016	Klausureinsicht
	21.09.2016	Klausur PZR2 (Mi, 21.09. 12:00 Uhr , Raum B101, H Gauß)

■ Zeitplan Zug 2, Mittwoch (vorläufig)

	Datum	Thema
1	06.04.2016	
2	13.04.2016	Einführung, Ziele, Ablauf, Benotung
3	20.04.2016	Client-Server Architektur
4	27.04.2016	REST-APIs
5	04.05.2016	REST in node.js
6	11.05.2016	Debugging und Testen
7	18.05.2016	Strukturierung, Modularisierung
8	25.05.2016	Vertiefung einzelner Themen
9	01.06.2016	Datenhaltung, SQL, NoSql, primär mit MongoDB
10	08.06.2016	backbone.js als Gegenpart zu REST/node
11	15.06.2016	Authentifizierung und Patterns
12	22.06.2016	Mobile Development/Cross-Plattform-Development
13	29.06.2016	Gastdozent(en)
14	06.07.2016	Klausurvorbereitung
15	12.07.2016	Klausur PZR1 (Di, 12.07. 14:00 Uhr , Ingeborg-Meising-S.)
16	20.07.2016	Klausureinsicht
	21.09.2016	Klausur PZR2 (Mi, 21.09. 12:00 Uhr , Raum B101, H Gauß)

Agenda

- Wiederholung
- Update zum Semesterplan
- Nachtrag: REST Schnittstellen generieren, testen, dokumentieren

- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
 - Requires
 - Middleware: Unterschied `.use()`, `.get()`
 - Request und Response
 - Parameter auslesen
 - Fehlerbehandlung
- Code-Übung zum Aufbau der API

- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon

- Zusammenfassende Fragen
- Ausblick

REST Schnittstellen Generieren, Testen, Dokumentieren

■ Generieren

- REST Beschreibung
→ Code Generator
- Frameworks für Annotationen
- Frameworks für REST-Layer

■ Testen

- Client-seitig
- Entwickler-Tools
(Browser)
- Teilautomatische Tests

■ Dokumentieren

- REST Beschreibung
→ Code Generator
- Frameworks für
Annotationen
- Frameworks für
REST-Layer

REST Testen

- **Apigee.com/console**
 - Für viele große US-Anbieter auch Parameter bereits mit dokumentiert

The screenshot displays the Apigee console interface for testing the Instagram API. The top navigation bar includes 'Providers', 'API Resources', and 'Console'. The 'Console' tab is selected, showing a 'Request URL' section with a GET request to the Instagram media search endpoint. Below this, the 'Request' and 'Response' sections are visible. The 'Request' section shows the raw HTTP request, and the 'Response' section shows the raw HTTP response, including status code 200 OK and various headers. A 'Snapshot' button is also present in the response section.

```
GET /v1/media/search?
access_token=2249200122.1fb234f.d23576f971c0491
HTTP/1.1
Host: api.instagram.com
X-Target-URI: https://api.instagram.com
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Content-Language: en
X-Ratelimit-Limit: 5000
Vary: Cookie, Accept-Language, Accept-Encoding
Date: Sat, 24 Oct 2015 20:57:20 GMT
Content-Length: 42701
Expires: Sat, 01 Jan 2000 00:00:00 GMT
X-Ratelimit-Remaining: 4999
Set-Cookie:
csrfToken=462067a6b72bb4b6f25414fd55df7f15;
expires=Sat, 22-Oct-2016 20:57:20 GMT; Max-
Age=31449600; Path=/
Connection: keep-alive
Content-Type: application/json; charset=utf-8
Server: Apigee Router
Cache-Control: private, no-cache, no-store, must-
revalidate
Pragma: no-cache
```

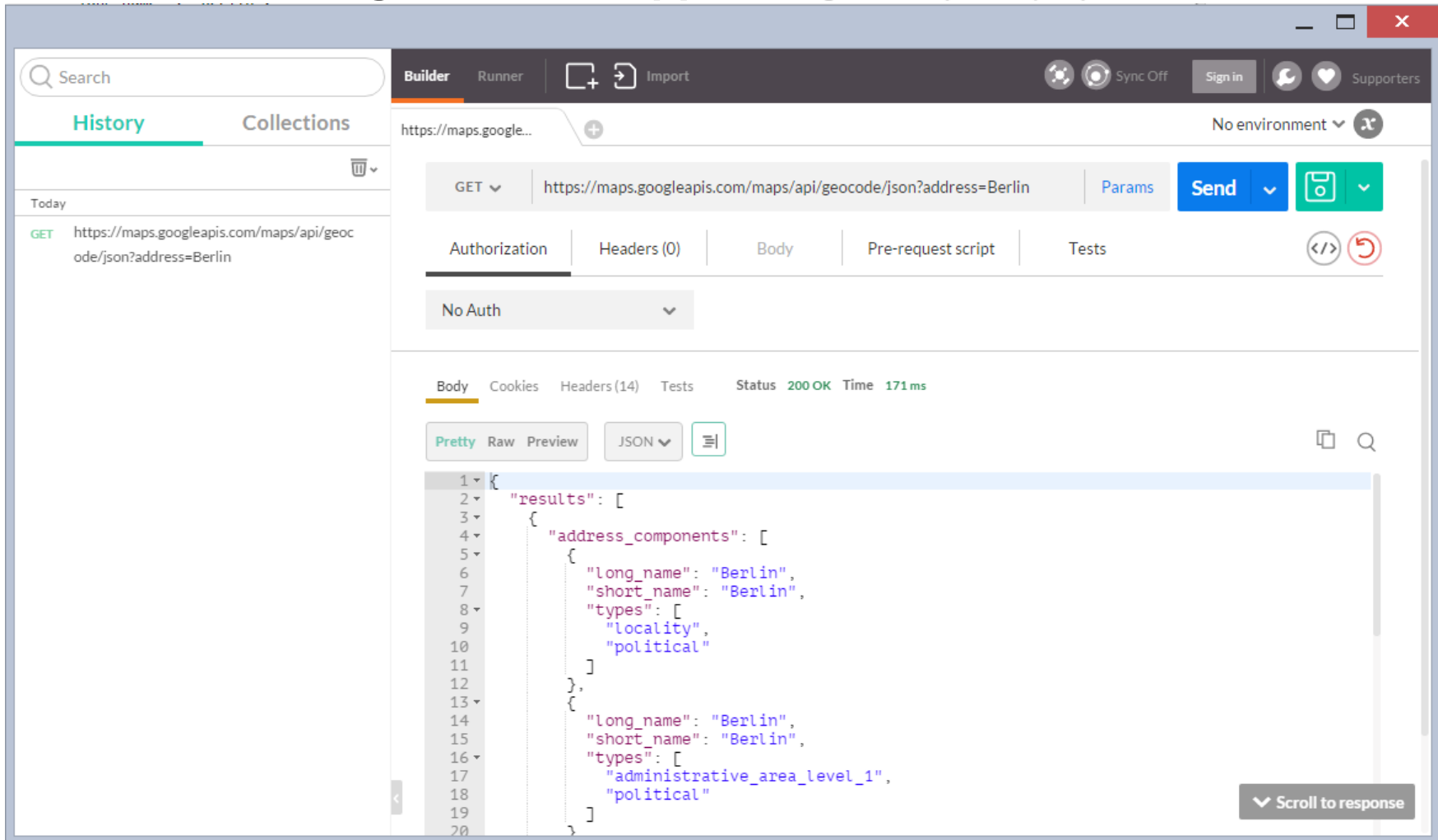
```
{
  "meta": {
    "code": 200
  },
  "data": [
    {
      "attribution": null,
      "tags": [],
      "type": "image",
    }
  ]
}
```

REST Testen

■ Postman Google Chrome App

Welcome to Postman 3.0

A great new experience, jam-packed with features



The screenshot displays the Postman 3.0 interface within a Google Chrome browser window. The left sidebar shows the 'History' tab with a single entry for a GET request to `https://maps.googleapis.com/maps/api/geocode/json?address=Berlin`. The main panel shows the 'Builder' tab for this request. The URL is `https://maps.googleapis.com/maps/api/geocode/json?address=Berlin`. The 'Send' button is visible. Below the URL bar, the 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is also visible. The response is displayed in the 'Body' tab, showing a JSON object with 'results' containing two entries for Berlin. The status is '200 OK' and the time is '171 ms'. The response is formatted as 'Pretty' JSON.

```
1 {
2   "results": [
3     {
4       "address_components": [
5         {
6           "long_name": "Berlin",
7           "short_name": "Berlin",
8           "types": [
9             "locality",
10            "political"
11          ]
12        },
13        {
14          "long_name": "Berlin",
15          "short_name": "Berlin",
16          "types": [
17            "administrative_area_level_1",
18            "political"
19          ]
20        }
21      ]
22    }
23  ]
24 }
```

REST Testen

- **Frisby on Jasmine oder Mocha für REST**

- **Test-Case Definitionen
(so wie bei JUnit-Tests)**

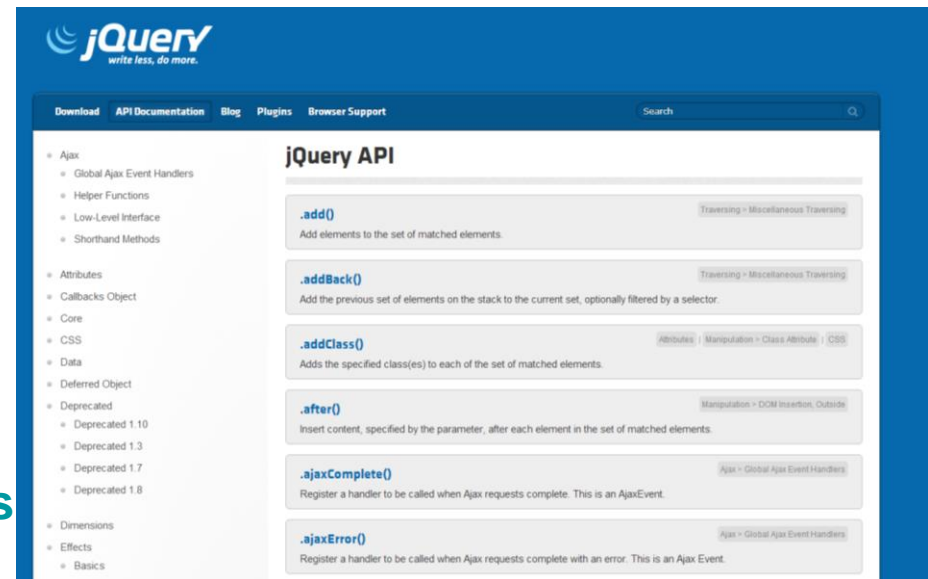


- **Werden wir im 4. Übungsblatt nutzen**

```
describe('Clean /video REST API ', function() {  
  it('should send status 204 and empty body ', function (done) {  
    request(videoURL)  
      .get('/')  
      .set('Accept-Version', '1.0')  
      .set('Accept', 'application/json')  
      .expect(codes.nocontent)  
      .end(function (err, res) {  
        should.not.exist(err);  
        res.body.should.be.empty();  
        done();  
      })  
  });  
});
```

REST Dokumentieren

- **Manuell (Word, LaTeX, ...)**
 - Für kleine APIs passend
- **Per Framework integriert generieren**
 - Bspw. Restler3 für PHP (s.u.)
- **Swagger.io (s.u.)**
 - Für mittlere APIs und
 - Unterstützte Code-Frameworks (node.js, PHP, .. JAX-RS)



REST Generieren (und Dokumentieren)

- Mit RESTLER 3



- PHP Rest Framework

- Einfache OAuth2 Integration
- Annotationen in PHPDoc
 - Automatische API Dokumentation

REST Generieren (und Dokumentieren)

- **RESTLER nutzt die Annotationen**
(Hier am Beispiel der Klasse Authors, Methode GET)





```
/**  
 * @param int $id  
 *  
 * @return array  
 */  
function get($id)  
{  
    $r = $this->dp->get($id);  
    if ($r === false)  
        throw new RestException(404);  
    return $r;  
}
```

REST Generieren (und Dokumentieren)

- **RESTLER nutzt die Annotationen..und generiert eine Doku**

/authors

Show/Hide | List Operations | Expand Operations | Raw

GET	/authors.json	routes to <code>improved\Authors::index();</code> 
GET	/authors.json/{id}	routes to <code>improved\Authors::get();</code> 
POST	/authors.json	routes to <code>improved\Authors::post();</code> 
PUT	/authors.json/{id}	routes to <code>improved\Authors::put();</code> 
PATCH	/authors.json/{id}	routes to <code>improved\Authors::patch();</code> 
DELETE	/authors.json/{id}	routes to <code>improved\Authors::delete();</code> 

REST Generieren (und Dokumentieren)

- **Swagger.io Editor**
- **Design->CodeGen**
- **API-Code->Doku**

The screenshot shows the Swagger.io Editor interface. On the left, the Swagger JSON definition is displayed in a code editor. On the right, the generated API documentation is shown, including the title 'Uber API', version '1.0.0', and a list of paths. The 'GET /products' path is selected, showing its summary, description, and parameters.

```

1  swagger: '2.0'
2  info:
3    title: Uber API
4    description: Move your app forward with the Uber API
5    version: 1.0.0
6    host: api.uber.com
7  schemes:
8    - https
9  basePath: /v1
10 produces:
11   - application/json
12 paths:
13   /products:
14     get:
15       summary: Product Types
16       description: |
17         The Products endpoint returns information about the
18         *Uber* products
19         offered at a given location. The response includes
20         the display name
21         and other details about each product, and lists the
22         products in the
23         proper display order.
24       parameters:
25         - name: latitude
26           in: query
27           description: Latitude component of location.
28           required: true
29           type: number
30           format: double
31         - name: longitude
32           in: query
33           description: Longitude component of location.
34           required: true
35           type: number
36           format: double
37       tags:
38         - Products
39       responses:
40         '200':
41           description: An array of products
42           schema:
43             type: array
44             items:
45               $ref: '#/definitions/Product'
46         default:
47           description: Unexpected error
48           schema:
49             $ref: '#/definitions/Error'
50   /estimates/price:
51     get:
52       summary: Price Estimates
53       description: >
54         The Price Estimates endpoint returns information about the
55         *Uber* products offered at a given location. The response includes
56         the display name and other details about each product, and lists the
57         products in the proper display order.

```

Uber API
Move your app forward with the Uber API
Version 1.0.0
Filter operations by a tag: Products Estimates User

Paths
/products

GET /products Products

Summary
Product Types

Description
The Products endpoint returns information about the *Uber* products offered at a given location. The response includes the display name and other details about each product, and lists the products in the proper display order.

Parameters

Name	Located in	Description	Required	Schema
latitude	query	Latitude component of location.	Yes	number (double)
longitude	query	Longitude component of location.	Yes	number (double)

Responses

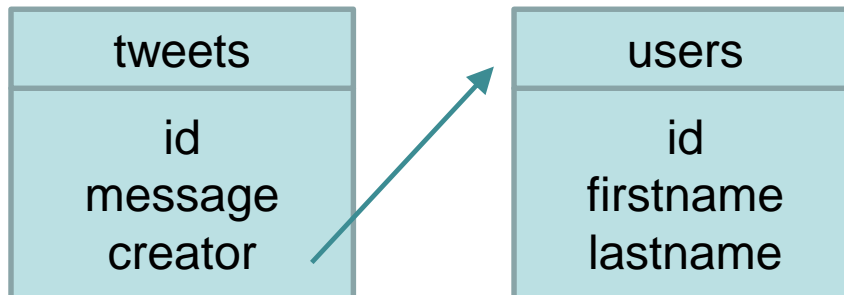
Agenda

- Wiederholung
 - Update zum Semesterplan
 - Nachtrag: REST Schnittstellen generieren, testen, dokumentieren
-
- Ziel: TwitterApp REST-API
 - Aufbau einer nodejs Anwendung
 - Requires
 - Middleware: Unterschied `.use()`, `.get()`
 - Request und Response
 - Parameter auslesen
 - Fehlerbehandlung
 - Code-Übung zum Aufbau der API
 - Kapselung von Routes mit `express.Router()`
 - Postman im Einsatz
 - Modul nodemon
 - Zusammenfassende Fragen
 - Ausblick

Beispielszenario als „Ziel“

- **Ihr Kunde möchte eine Art kleines Twitter für den firmeninternen Gebrauch haben.** Dazu konzentrieren Sie sich zunächst auf die REST-API und möchten die (diversen) Clients dafür später entwickeln.
- **Technologie:** nodejs/express. Ihr Kunde wünscht explizit, dass keine module wie node-restful oder ähnliches verwendet werden.

- **Entities**



- **Fokus: erstmal die Tweets**

Beispielszenario miniTwitter-REST API

Unsere TODOs (grob)

- URLs „matchen“ (für Ressourcen)
- HTTP Methoden bedienen (für Operationen)
- Type prüfen (für Repräsentation)

- Parameter auslesen (JSON aus Body bei POST)
- Store anbinden
- Fehler behandeln
- HTTP Status-Codes verwenden

- Testen
- Dokumentieren

Agenda

- Wiederholung
- Update zum Semesterplan
- Nachtrag: REST Schnittstellen generieren, testen, dokumentieren

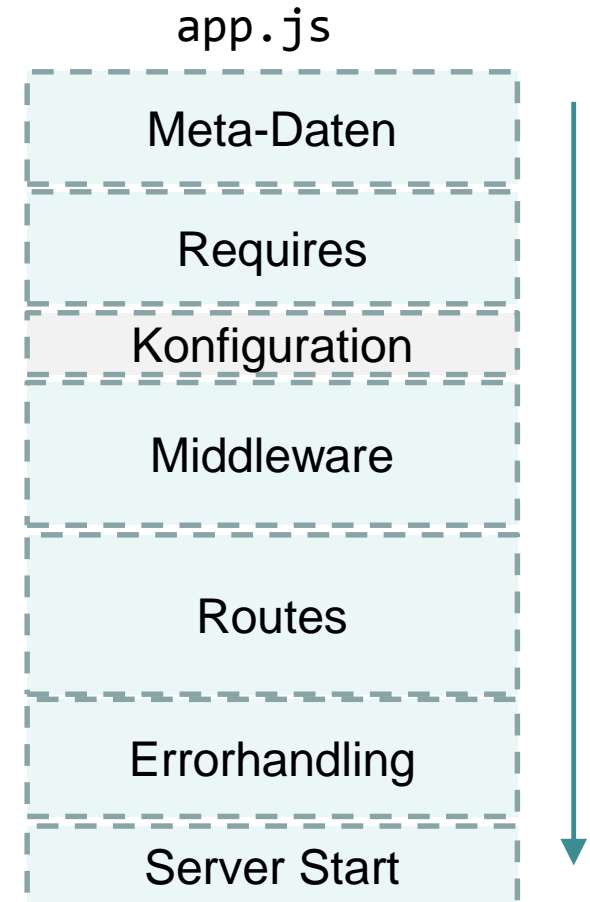
- Ziel: TwitterApp REST-API
- **Aufbau einer nodejs Anwendung**
 - Requires
 - Middleware: Unterschied `.use()`, `.get()`
 - Request und Response
 - Parameter auslesen
 - Fehlerbehandlung
- Code-Übung zum Aufbau der API

- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon

- Zusammenfassende Fragen
- Ausblick

Node.js Aufbau einer Server-Anwendung (Big Picture)

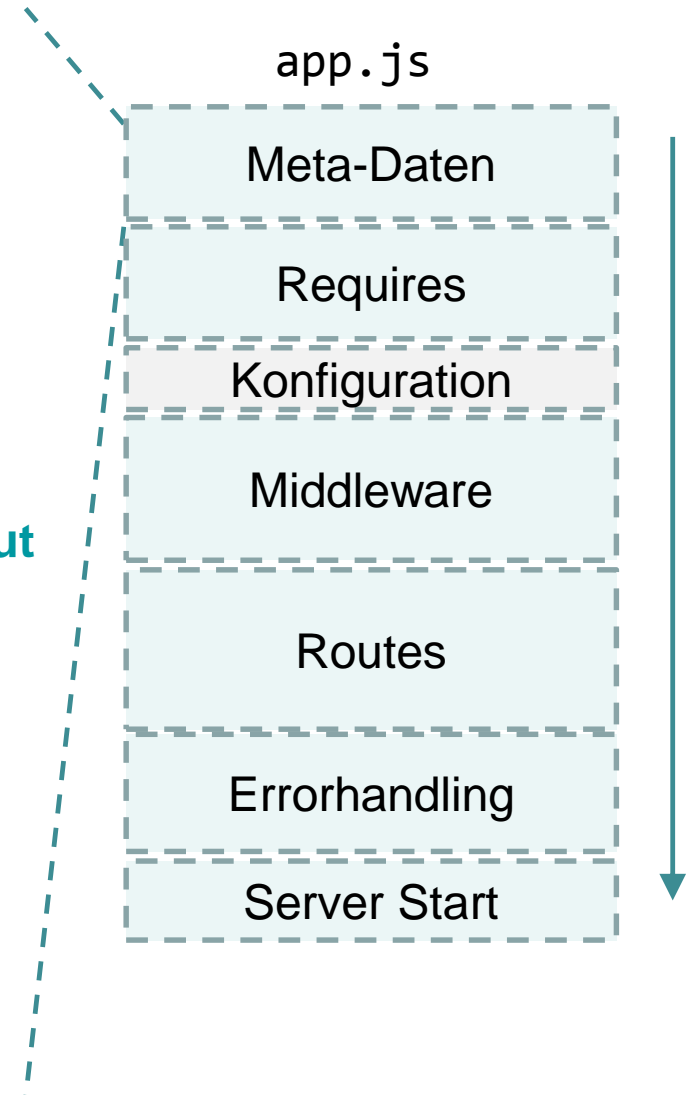
- **Konzept:**
 - von oben nach unten
 - erst Handler registrieren
 - dann Starten



Node.js Aufbau einer Server-Anwendung (Details)

```
/** Main app for server to start ...bla
 *
 *
 * @author Johannes Konert
 * @licence CC BY-SA 4.0
 *
 */
"use strict";
```

- Beschreiben Sie,
 - was ihre Hauptanwendung (od. Modul) tut
 - geben Sie mindestens einen Autor an
 - (von E-Mail Addr. rate ich ab)
 - ggf. Lizenzmodell usw
 - siehe bspw. JSDoc <http://usejsdoc.org/>



Agenda

- Wiederholung
- Update zum Semesterplan
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung

Requires

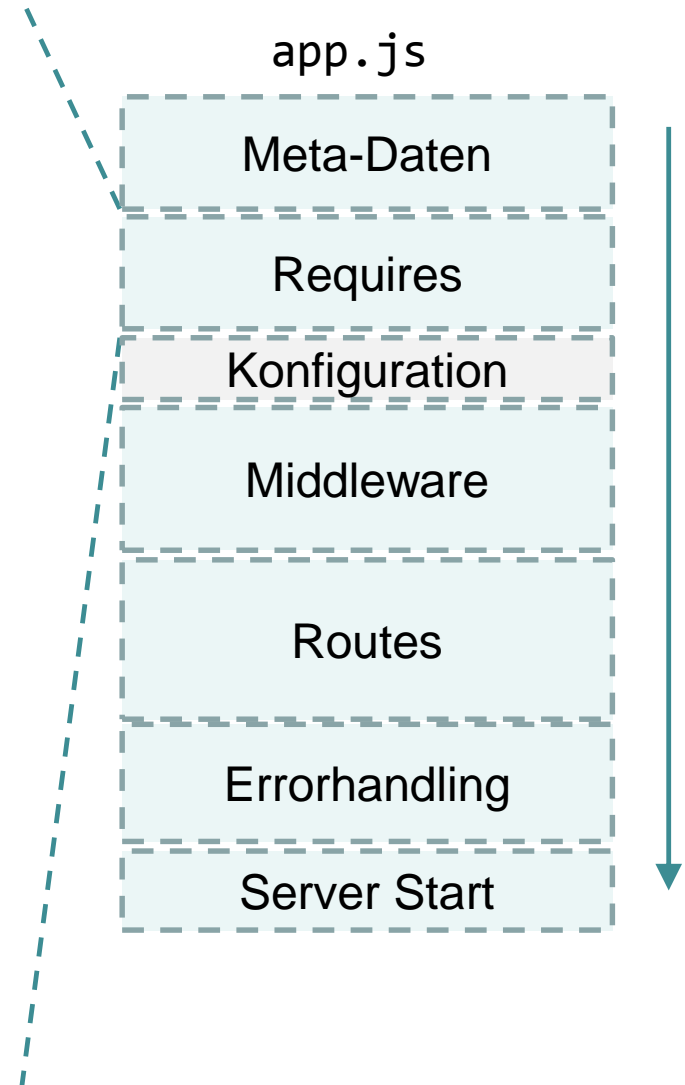
- Middleware: Unterschied `.use()`, `.get()`
- Request und Response
- Parameter auslesen
- Fehlerbehandlung
- Code-Übung zum Aufbau der API
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
- Zusammenfassende Fragen
- Ausblick

Node.js Aufbau einer Server-Anwendung (Details)

```
// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');

// own modules imports
var store = require('./blackbox/store.js');
```

- Erst node.js **interne** Module
(brauchen kein npm install)
- dann **installierte** Module
(wurden npm installiert)
- dann **eigene** JS-Dateien
(relative Pfade)



Node.js Aufbau einer Server-Anwendung (Details)

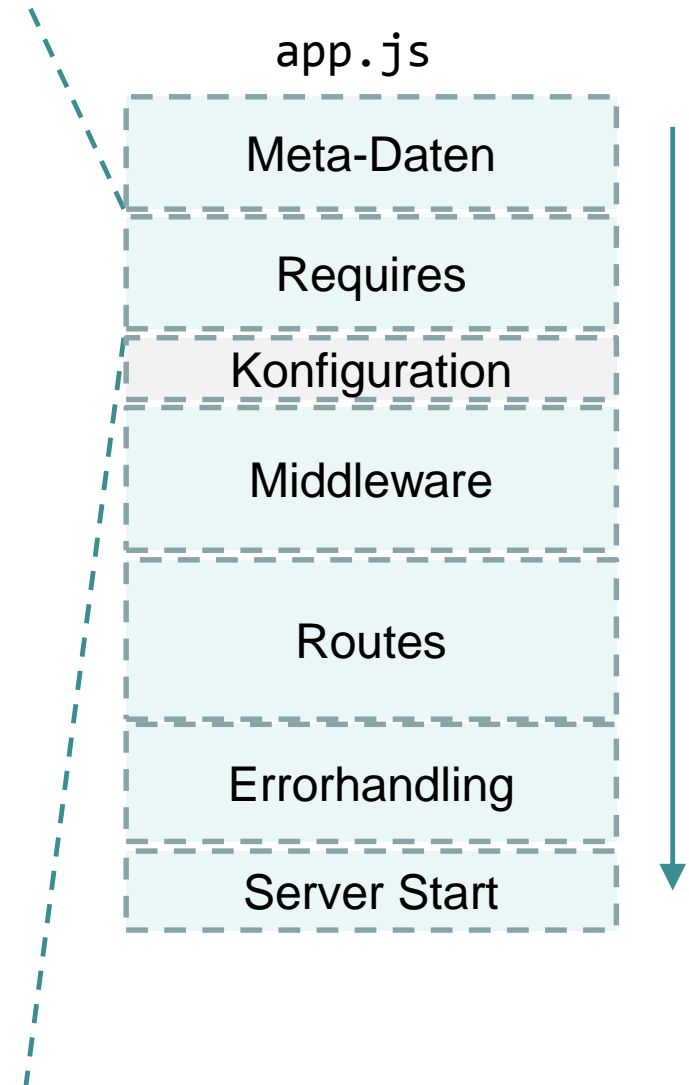
```
// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');

// own modules imports
var store = require('./blackbox/store.js');
```

Modul body-parser bietet

- JSON body parser
- Raw body parser
- Text body parser
- URL-encoded form body parser

für: gesendete JSON Daten im POST oder
gesendete Formulardaten parsen.



Node.js Aufbau einer Server-Anwendung (Details)

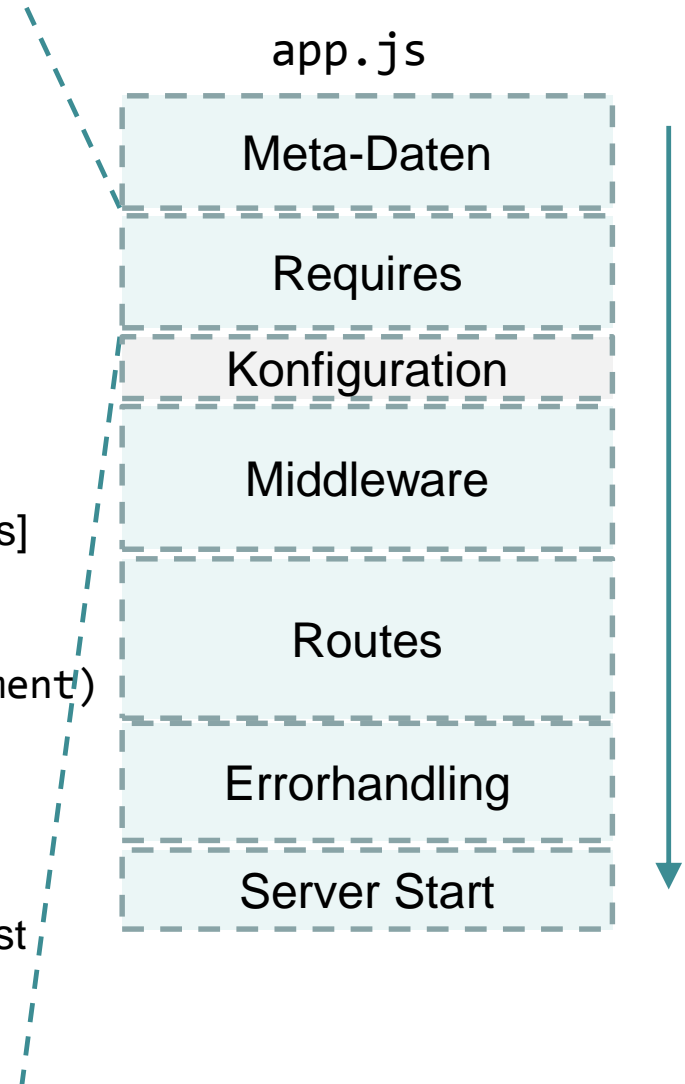
```
// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');
```

```
// own modules imports
var store = require('./blackbox/store.js');
```

Modul **store** bietet vier Methoden**

- **select** (**String** type, **Number** id)
[@returns undefined, one element or array of elements]
- **insert** (**String** type, **Object** element)
[@returns ID of new element]
- **replace** (**String** type, **Number** id, **Object** element)
[@returns this (the store object)]
- **remove** (**String** type, **Number** id)
[@returns this (the store object)]

für: Dummy Speicherung von Objekten, damit Sie zunächst keine DB brauchen.



Agenda

- Wiederholung
- Update zum Semesterplan

- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
 - Requires
- Middleware: Unterschied `.use()`, `.get()`
 - Request und Response
 - Parameter auslesen
 - Fehlerbehandlung
- Code-Übung zum Aufbau der API

- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon

- Zusammenfassende Fragen
- Ausblick

Node.js Aufbau einer Server-Anwendung (Details)

- typischerweise die meisten `app.use(...)` ;
- Unterschied `app.use(...)`, `app.get(...)` ?

`.use(prefix, handler)`

- trifft auf alle HTTP Methoden zu
- prefix ist ein (optionaler) URL-Route Präfix
- Handler ist Function

`.get(pattern, handler)`

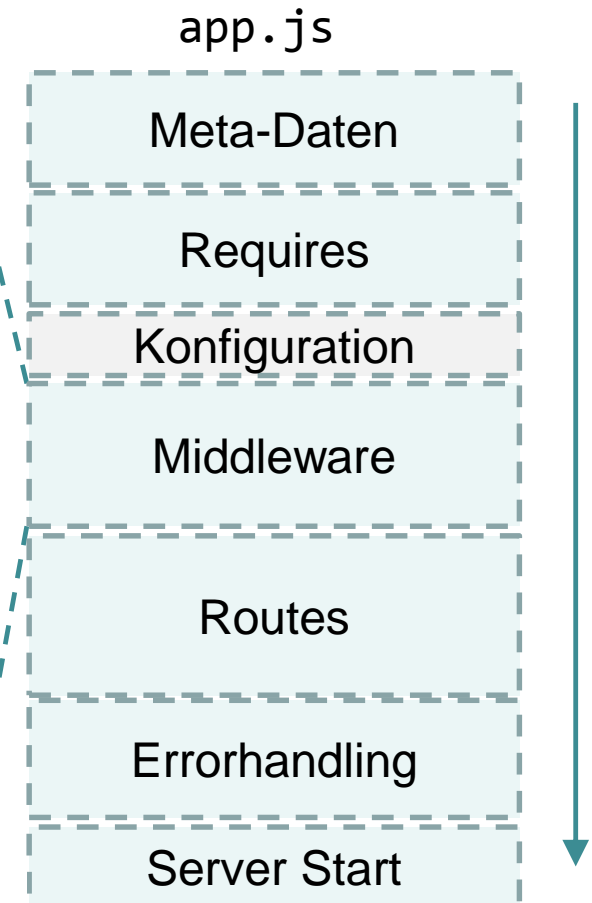
- trifft auf GET HTTP Methode zu
- pattern ist ein Muster (String oder RegExp)
- Handler ist Function

```
app.use('/', function(...) {...})
```

```
app.get('/', function(...) {...})
```

```
app.post('/', function(...) {...})
```

```
app.all('/', function(...) {...}) = alle HTTP Methoden!
```

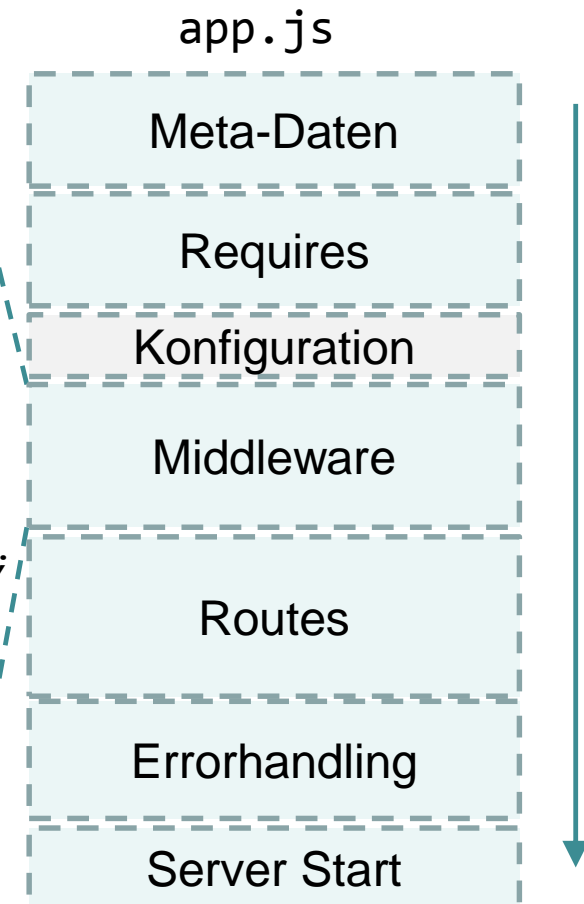


Node.js Aufbau einer Server-Anwendung (Details)

- typischerweise die meisten `app.use (...)` ;
- Middleware für
 - Authentifizierung
 - Daten anhand von Parametern bereits laden
 - Logging
- ...dann Request „weiterreichen“

```
app.use(function(req, res, next) {  
  console.log('Request of type ' + req.method  
              + ' to URL ' + req.originalUrl);  
  next();  
});
```

- Geben Sie drei Parameter an, um den next() Handler aufrufen zu können!
- DesignPattern?
 - Das ist das **Chain-of-Responsibility Pattern**
 - Ziel: Lose Kopplung
 - inkl. Dependency Injection



Agenda

- Wiederholung
- Update zum Semesterplan

- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
 - Requires
 - Middleware: Unterschied `.use()`, `.get()`
- Request und Response
- Parameter auslesen
- Fehlerbehandlung
- Code-Übung zum Aufbau der API

- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon

- Zusammenfassende Fragen
- Ausblick

Node.js Aufbau einer Server-Anwendung (Details)

■ Request-Objekt bietet viele Informationen zur Anfrage

- Header-Angaben

```
req.get('Accept-Version')
```



```
req.get('Content-Type')
```

 - inkl. Convenience Methoden

```
req.accepts('json')
```

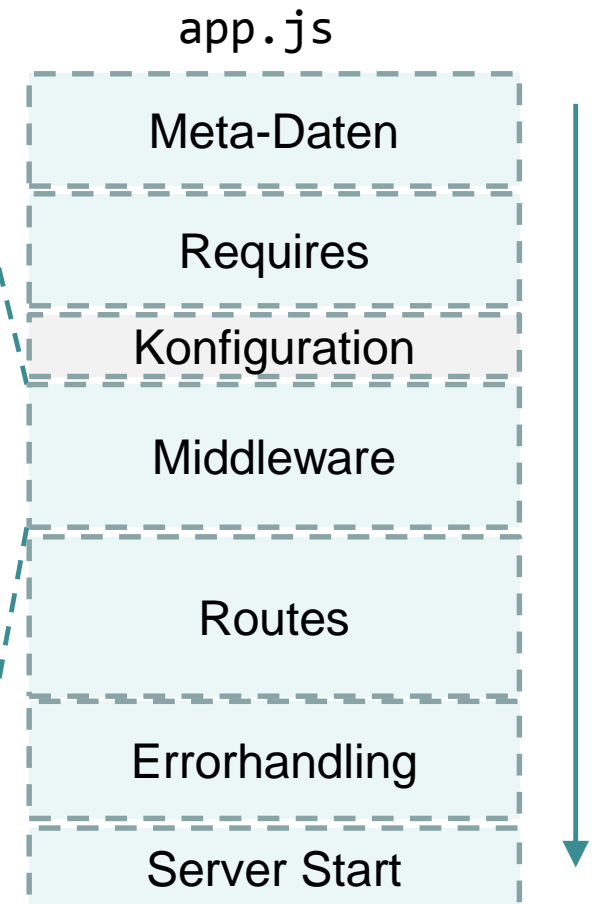
■ Response-Objekt ebenfalls

- ```
res.set('Content-Type', 'text/plain');
```
- direkt ein Objekt in JSON umwandeln und als Content-Type JSON senden 

```
res.json(element)
```
  - -Type u. - Length autom. 

```
res.send('<!DOC...')
```
  - Response unterstützt Verkettung \*\*  

```
res.status(200).end();
```
  - Design-Pattern?
    - **Nein**, das ist **Konzept des Fluent Interfaces**  
(realisiert mit „return this“ in jeder Methode)



## Node.js Aufbau einer Server-Anwendung (Details)

- **Routes enthält** dann die verschiedenen **URL-Handler für Ressourcensammlungen**

```
app.get('/tweets/:id', function(req, res, next) {
 ... req.params.id;
});
```

- mit `:` können Url-Teile als Variable definiert werden und sind dann in `req.params` verfügbar

- GET-Parameter (die `?key=value`)  
zum Beispiel `?offset=12`    `req.query.offset;`

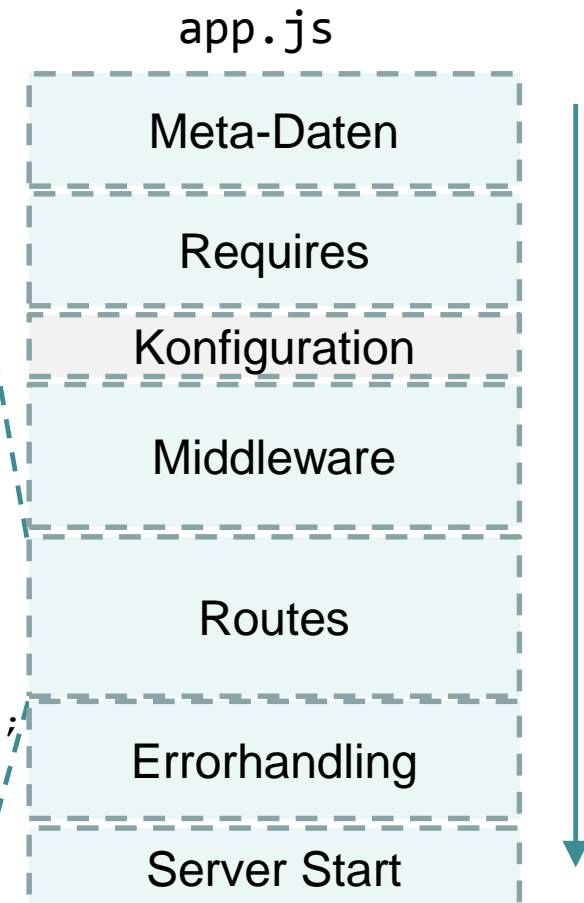
- POST-Parameter

(1) mit Middleware `body-parser`

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
```

(2) diese fügt die Parameter als Objekt in Request ein

```
var obj = req.body;
```



# Agenda

- Wiederholung
- Update zum Semesterplan
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
  - Requires
  - Middleware: Unterschied `.use()`, `.get()`
  - Request und Response
  - Parameter auslesen

## Fehlerbehandlung

- Code-Übung zum Aufbau der API
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
- Zusammenfassende Fragen
- Ausblick

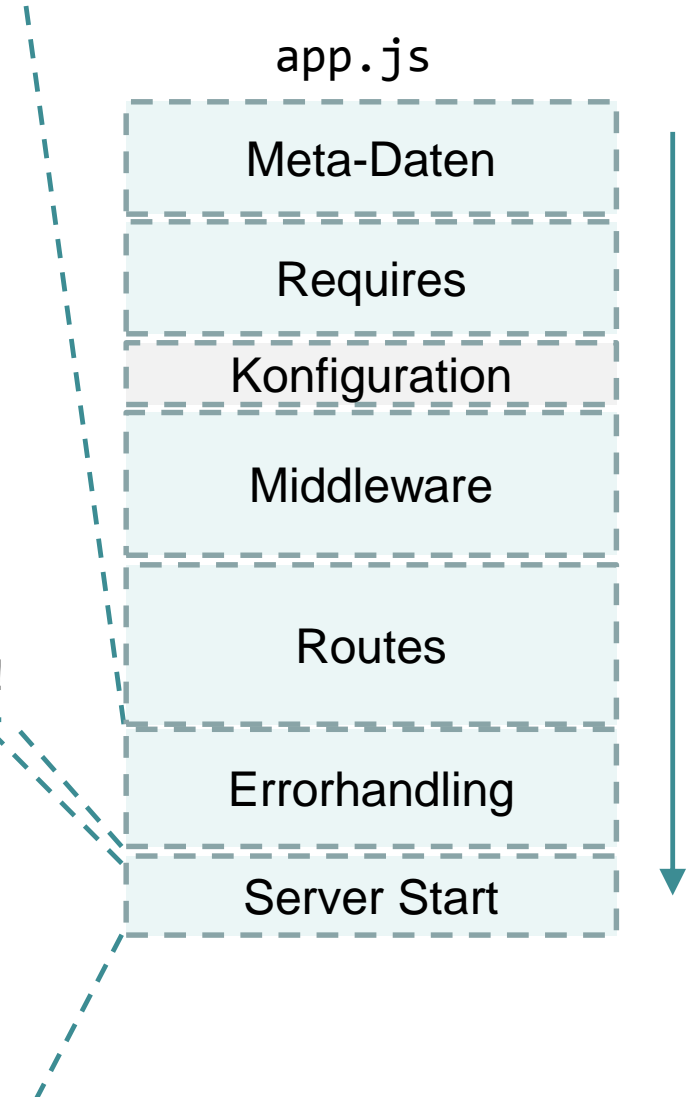
## Node.js Aufbau einer Server-Anwendung (Details)

- Eine Art Handler für die „Reste“ am Ende, denn **jeder** Request braucht eine Antwort

```
app.use(function(req, res, next) {
 var err = new Error('Not Found');
 err.status = 404;
 next(err);
});
...
app.use(function(err, req, res, next) {
 ...
 res.status(err.status).end();
});
```

- Express erkennt anhand der vier statt drei Parameter, dass diese Middleware nur bei Errors aufgerufen wird!

```
app.listen(3000, function(err) {
 if (err !== undefined) {
 console.log("Error on startup, ",err);
 }
 else {
 console.log("Listening on port 3000");
 }
});
```



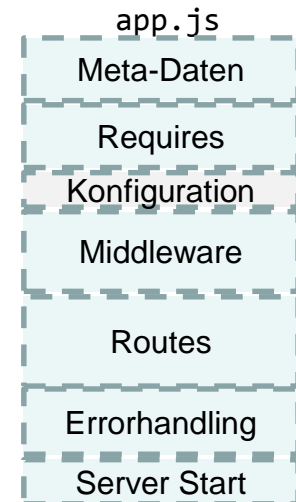
# Agenda

- Wiederholung
- Update zum Semesterplan
  
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
  - Requires
  - Middleware: Unterschied `.use()`, `.get()`
  - Request und Response
  - Parameter auslesen
  - Fehlerbehandlung
- Code-Übung zum Aufbau der API
  
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
  
- Zusammenfassende Fragen
- Ausblick

# Node.js Aufbau einer Server-Anwendung

## Aufgabe

- In welcher Reihenfolge müssen die Code-Bausteine zusammengefügt werden, damit eine Tweet-API in nodejs entsteht? (5min – max. 10min)
  - Sie erhalten 12 Code-Schnipsel pro Team
  - Teilen Sie die Code-Schnipsel gleichmäßig auf
  - Jeder: Lesen und verstehen der eigenen Code-Teile
  - Tauschen Sie im Team, wenn Code unverständlich ist
  - Ziel: Legen Sie gemeinsam eine Reihenfolge fest
  - Wenn Sie fertig sind, helfen Sie anderen Teams ohne die Lösung direkt zu verraten.
- Anschließend:  
Online-Abstimmung





# Node.js Aufbau einer Server-Anwendung

## Aufgabe

- Anschließend:  
Online-Abstimmung

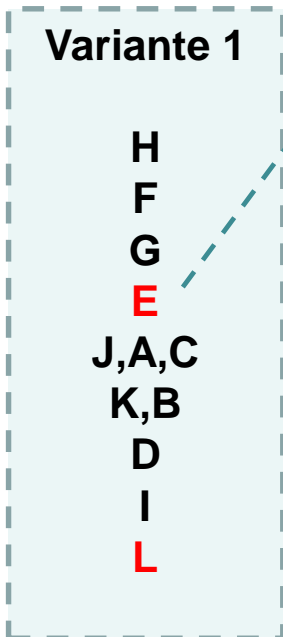


| Variante 1 | Variante 2 | Variante 3 | Variante 4 | Variante 5      |               |
|------------|------------|------------|------------|-----------------|---------------|
| H          | H          | H          | H          | <b>Sonstige</b> | app.js        |
| F          | F          | F          | F          |                 | Meta-Daten    |
| G          | G          | G          | G          |                 | Requires      |
| E          | J,L        | J,L        | J,A,C      |                 | Konfiguration |
| J,A,C      | D          | A,C        | K,B        |                 | Middleware    |
| K,B        | I          | K,B        | D          |                 | Routes        |
| D          | A,C        | D          | I          |                 | Errorhandling |
| I          | K,B        | I          | L          |                 | Server Start  |
| L          | E          | E          | E          |                 |               |

- Pingo URL: <http://pingo.upb.de/791474> (2min)

# Node.js Aufbau einer Server-Anwendung

## Lösungen

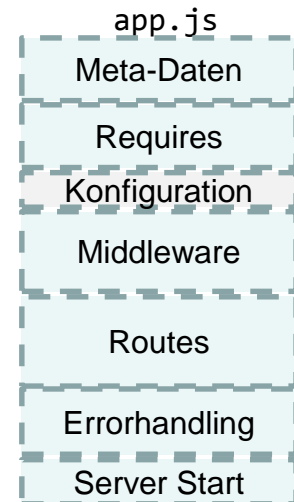


Server Start

`app.listen(..)` funktioniert auch früher

### Problem:

- Server nimmt bereits Anfragen entgegen
- Nicht alle Middleware und Routen sind registriert
- Potentielle Quelle für Fehler, Sicherheitslöcher und DB-Inkonsistenzen



Daher: Server erst „zuletzt“ starten, wenn alles konfiguriert und alle Handler bei `app.` registriert sind.

# Node.js Aufbau einer Server-Anwendung

## Lösungen



### Variante 1

H  
F  
G  
**E**  
J,A,C  
K,B  
D  
I  
**L**

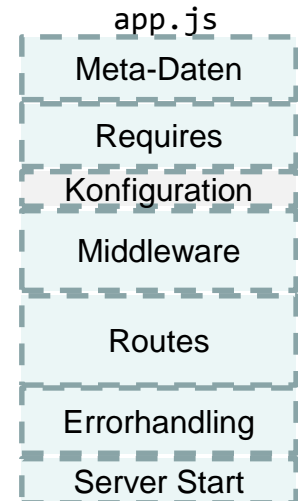
Logging  
Middleware

`app.use(...)` und `app.get(...)` etc.  
Reihenfolge entscheidend

### Problem:

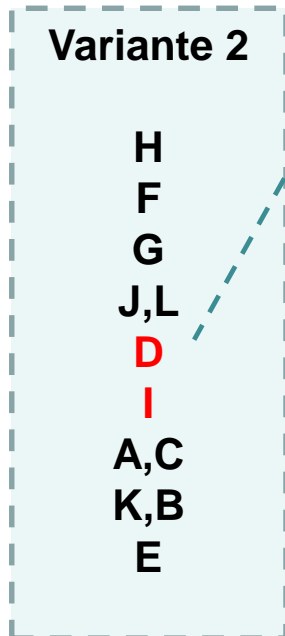
- Mit Logging am Ende der Kette registrierter Handler (Middleware und Routes) werden nur Anfragen geloggt, die
  - 1) auf keine vorherige Route zutreffen
  - 2) oder mit `next()` von vorherigem Handler weitergereicht werden

Daher: Logging besser so früh wie möglich einbauen (um alles mitzubekommen).



# Node.js Aufbau einer Server-Anwendung

## Lösungen



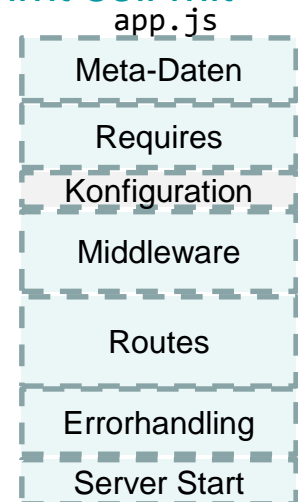
Errorhandling

Eine Art catchall Handler. Alles was bis zu ihm kommt soll mit einem Status 404 beantwortet werden.

Auch hier: Reihenfolge entscheidend

### Problem:

- Blöcke K und B mit den REST-Routen sollten vor D liegen, damit nur ungültige URLs mit 404 beantwortet werden



Daher: Middleware zum Abfangen von nicht existierenden URLs immer ans Ende nach den Handlern für die gültigen URLs.

# Node.js Aufbau einer Server-Anwendung

## Lösungen



### Variante 2

H  
F  
G  
J,L  
**D**  
I  
A,C  
K,B  
E

### Errorhandling

`app.use(function(err, req, res, next) { ... });`  
registriert Handler für Fehlerfälle.  
Wird nur aktiv bei `next(err);` Aufrufen.

### Problem?

- Wegen `err` als erstem Parameter eigentlich „egal“ wo diese Middleware registriert wird
- Zur Wartbarkeit und leichtem Codelesen besser hinter den normalen Routen und Middlewares angeben.

app.js  
Meta-Daten

Requires

Konfiguration

Middleware

Routes

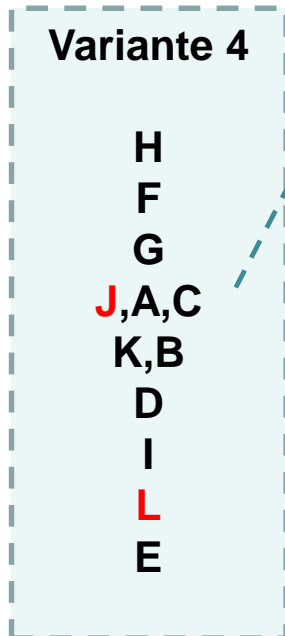
Errorhandling

Server Start

Daher besser: Fehlerbehandlung nach den erwarteten Routen und Fällen einfügen.

# Node.js Aufbau einer Server-Anwendung

## Lösungen

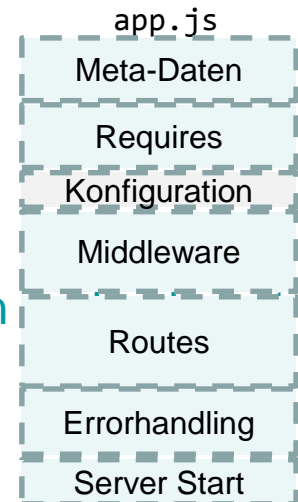


static files  
Middleware

Express.static Middleware zum Ausliefern statischer Dateien von der Festplatte

### Problem:

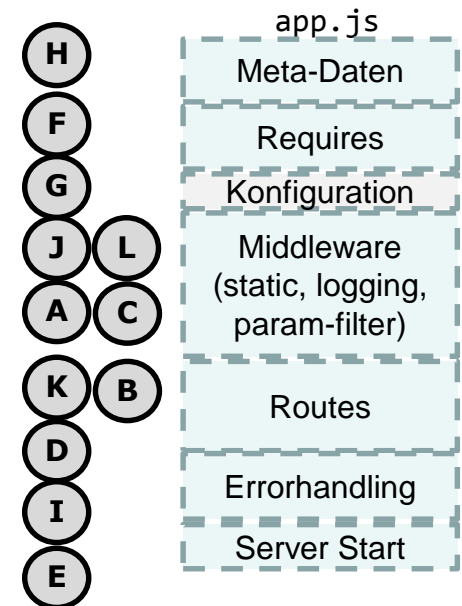
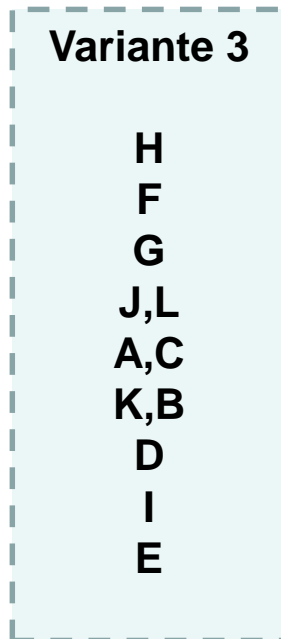
- Statische Dateien sollten vor Auswertung spezifischer Parameter der REST-API gefunden werden, sonst muss auch für bspw. index.html die Version und HTTP-Methode stimmen (A,C)



Daher: Prüfung auf Anfrage einer statische Datei so früh wie möglich als erste Middleware registrieren; dann sind diese Anfragen schon mal behandelt.

# Node.js Aufbau einer Server-Anwendung: Lösung

## ■ Lösung



# Agenda

- Wiederholung
  - Update zum Semesterplan
  
  - Ziel: TwitterApp REST-API
  - Aufbau einer nodejs Anwendung
    - Requires
    - Middleware: Unterschied `.use()`, `.get()`
    - Request und Response
    - Parameter auslesen
    - Fehlerbehandlung
  - Code-Übung zum Aufbau der API
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
  - Modul nodemon
  
  - Zusammenfassende Fragen
  - Ausblick



## Node.js: Hierarchien nutzen

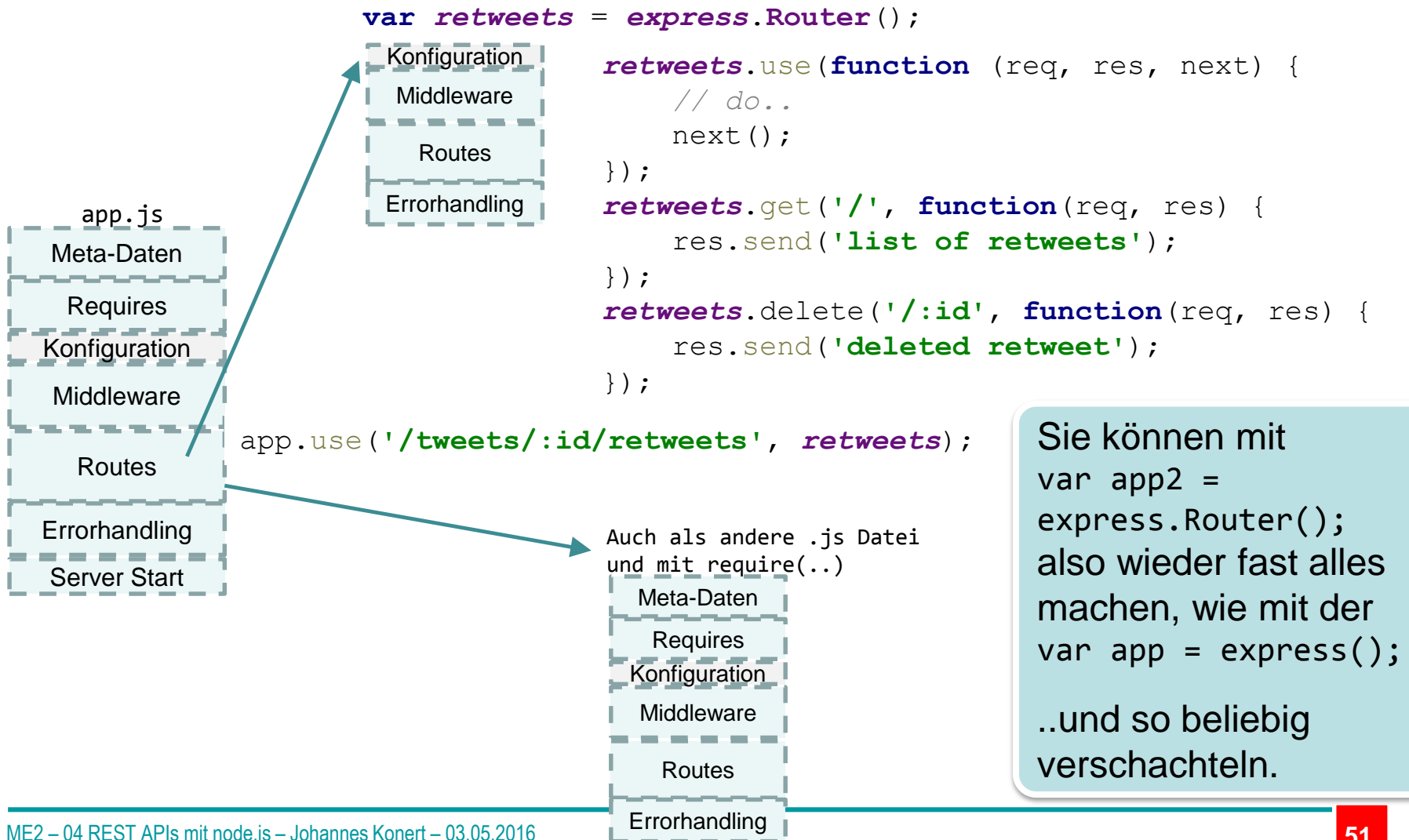
- **app.route()** erlaubt das Zusammenfassen verschiedener HTTP Methoden für gleiche Route

```
app.route('/tweets/:id')
 .get(function(req, res) {
 res.send('...');
 })
 .put(function(req, res) {
 res.send('...');
 })
 .delete(function(req, res) {
 res.send('...');
 });
```

- schon wieder: **Konzept des Fluent Interfaces**

# Node.js: Hierarchien nutzen

## Middlewares und Routes lassen sich verschachteln



# Agenda

- Wiederholung
- Update zum Semesterplan
  
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
  - Requires
  - Middleware: Unterschied `.use()`, `.get()`
  - Request und Response
  - Parameter auslesen
  - Fehlerbehandlung
- Code-Übung zum Aufbau der API
  
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
  
- Zusammenfassende Fragen
- Ausblick

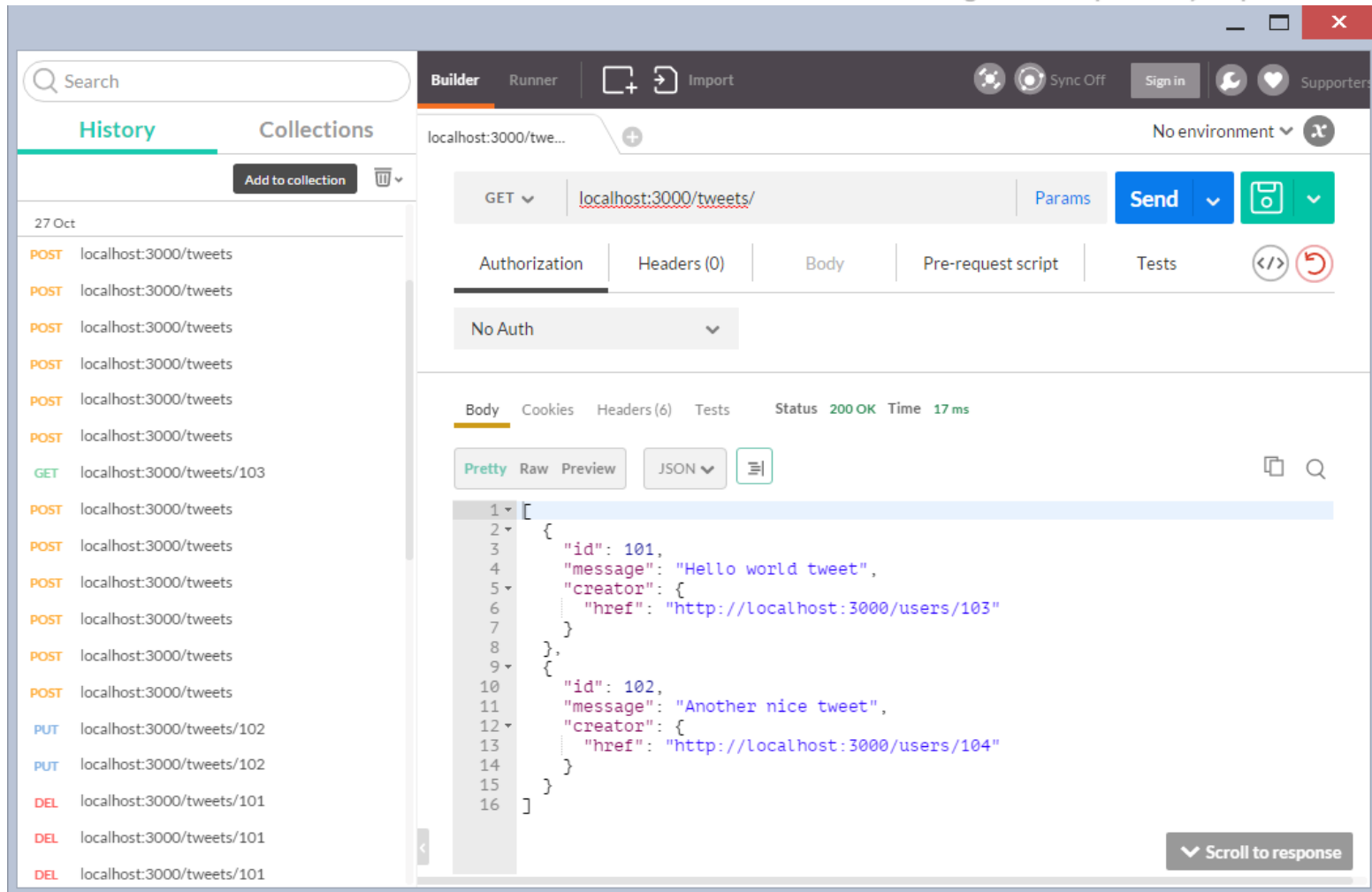
# Mit Postman GET, POST, PUT, DELETE

## ■ (Live Demo)



Welcome to Postman 3.0

A great new experience, jam-packed with features



The screenshot displays the Postman 3.0 interface. On the left, the 'History' tab shows a list of recent requests, including several POST requests to 'localhost:3000/tweets' and a GET request to 'localhost:3000/tweets/103'. The main panel shows a GET request to 'localhost:3000/tweets/'. The 'Send' button is highlighted. Below the request bar, the 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is also visible. The response is displayed in the bottom panel, showing a 200 OK status and a JSON array of two tweet objects. The first tweet has an id of 101 and a message of 'Hello world tweet'. The second tweet has an id of 102 and a message of 'Another nice tweet'.

```
[{"id": 101, "message": "Hello world tweet", "creator": {"href": "http://localhost:3000/users/103"}}, {"id": 102, "message": "Another nice tweet", "creator": {"href": "http://localhost:3000/users/104"}}]
```

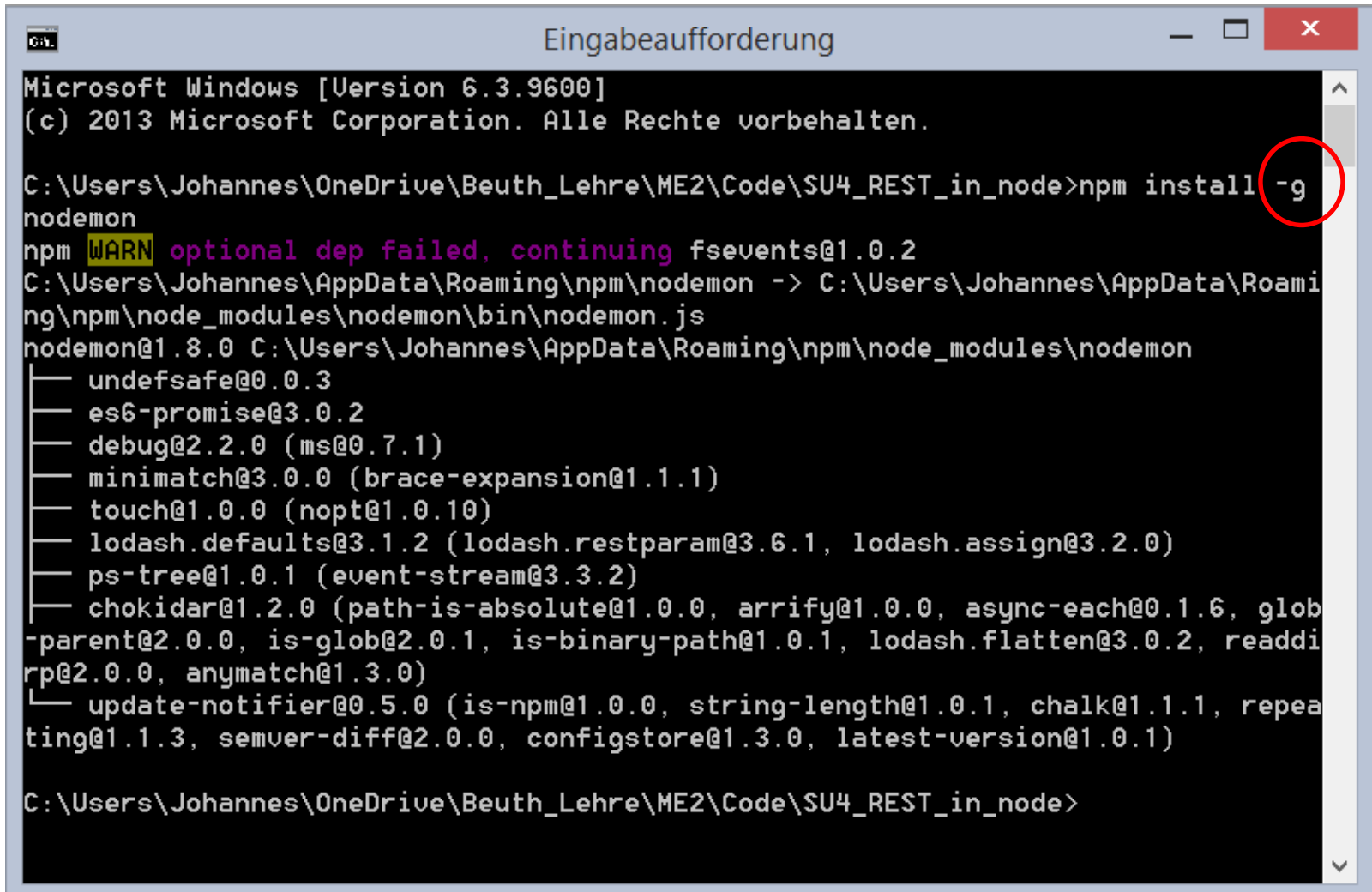
# Agenda

- Wiederholung
- Update zum Semesterplan
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
  - Requires
  - Middleware: Unterschied `.use()`, `.get()`
  - Request und Response
  - Parameter auslesen
  - Fehlerbehandlung
- Code-Übung zum Aufbau der API
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
- Zusammenfassende Fragen
- Ausblick

## Produktiv(er) entwickeln mit nodemon

- **Modul nodemon bietet**
  - Verwendung wie node zum Ausführen von bspw. app.js Code
  - Überwacht alle geladenen Ressourcen
  - Bei Änderung erfolgt automatischer Neustart
- **Installation?**
  - `npm install nodemon`

## Produktiv(er) entwickeln mit nodemon



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU4_REST_in_node>npm install -g
nodemon
npm WARN optional dep failed, continuing fsevents@1.0.2
C:\Users\Johannes\AppData\Roaming\npm\nodemon -> C:\Users\Johannes\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js
nodemon@1.8.0 C:\Users\Johannes\AppData\Roaming\npm\node_modules\nodemon
├── undefsafe@0.0.3
├── es6-promise@3.0.2
├── debug@2.2.0 (ms@0.7.1)
├── minimatch@3.0.0 (brace-expansion@1.1.1)
├── touch@1.0.0 (nopt@1.0.10)
├── lodash.defaults@3.1.2 (lodash.restparam@3.6.1, lodash.assign@3.2.0)
├── ps-tree@1.0.1 (event-stream@3.3.2)
├── chokidar@1.2.0 (path-is-absolute@1.0.0, arrify@1.0.0, async-each@0.1.6, glob
-parent@2.0.0, is-glob@2.0.1, is-binary-path@1.0.1, lodash.flatten@3.0.2, readdirp@2.0.0, anymatch@1.3.0)
├── update-notifier@0.5.0 (is-npm@1.0.0, string-length@1.0.1, chalk@1.1.1, repea
ting@1.1.3, semver-diff@2.0.0, configstore@1.3.0, latest-version@1.0.1)
C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU4_REST_in_node>
```

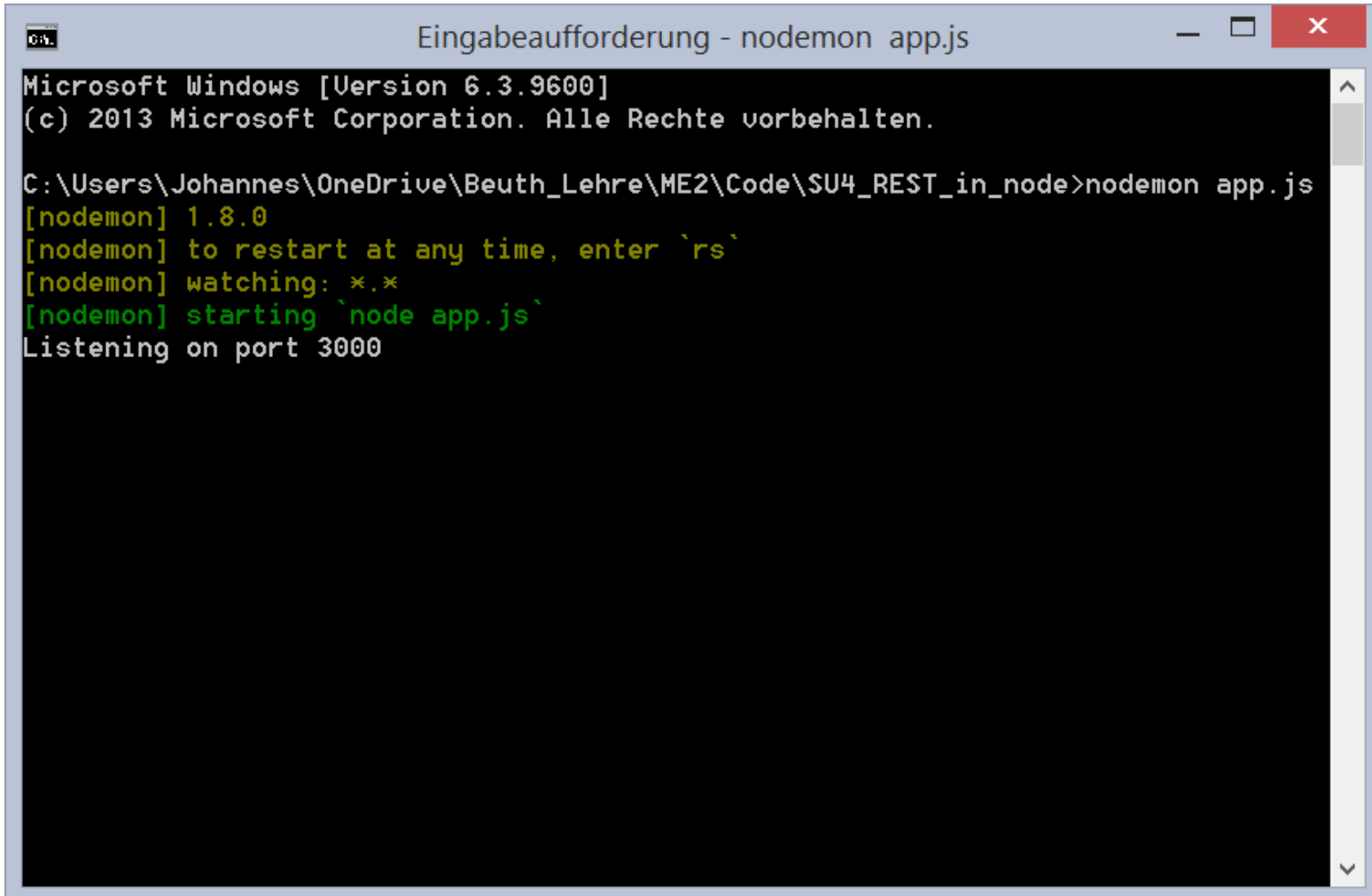
## Produktiv(er) entwickeln mit nodemon

- **Paketmanager npm erlaubt die Installation „global“, also ins nodejs Verzeichnis und ~nicht~ ins Projektverzeichnis**
  - **Gut für:** IDEs u. Werkzeuge, die nicht das ganze Dev-Team benutzt
  - **Schlecht für:** Module, die für das Testen und Betrieb des Projektes nötig sind
- `npm install -g <modulname>`



# Produktiv(er) entwickeln mit nodemon

## ■ nodemon app.js

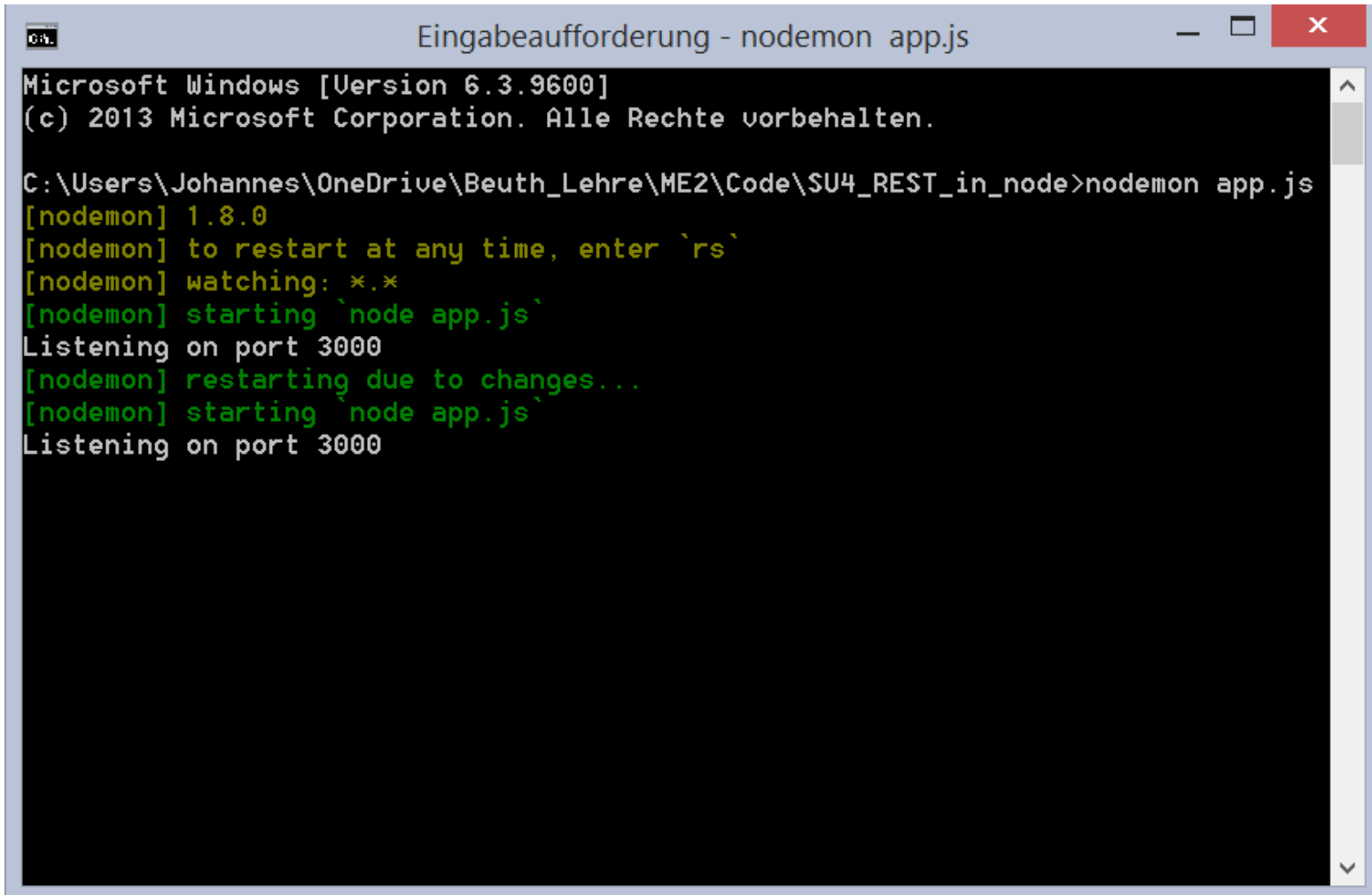


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU4_REST_in_node>nodemon app.js
[nodemon] 1.8.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Listening on port 3000
```

## Produktiv(er) entwickeln mit nodemon

- **nodemon app.js** (nach Änderung einer Datei automatisch Neustart)

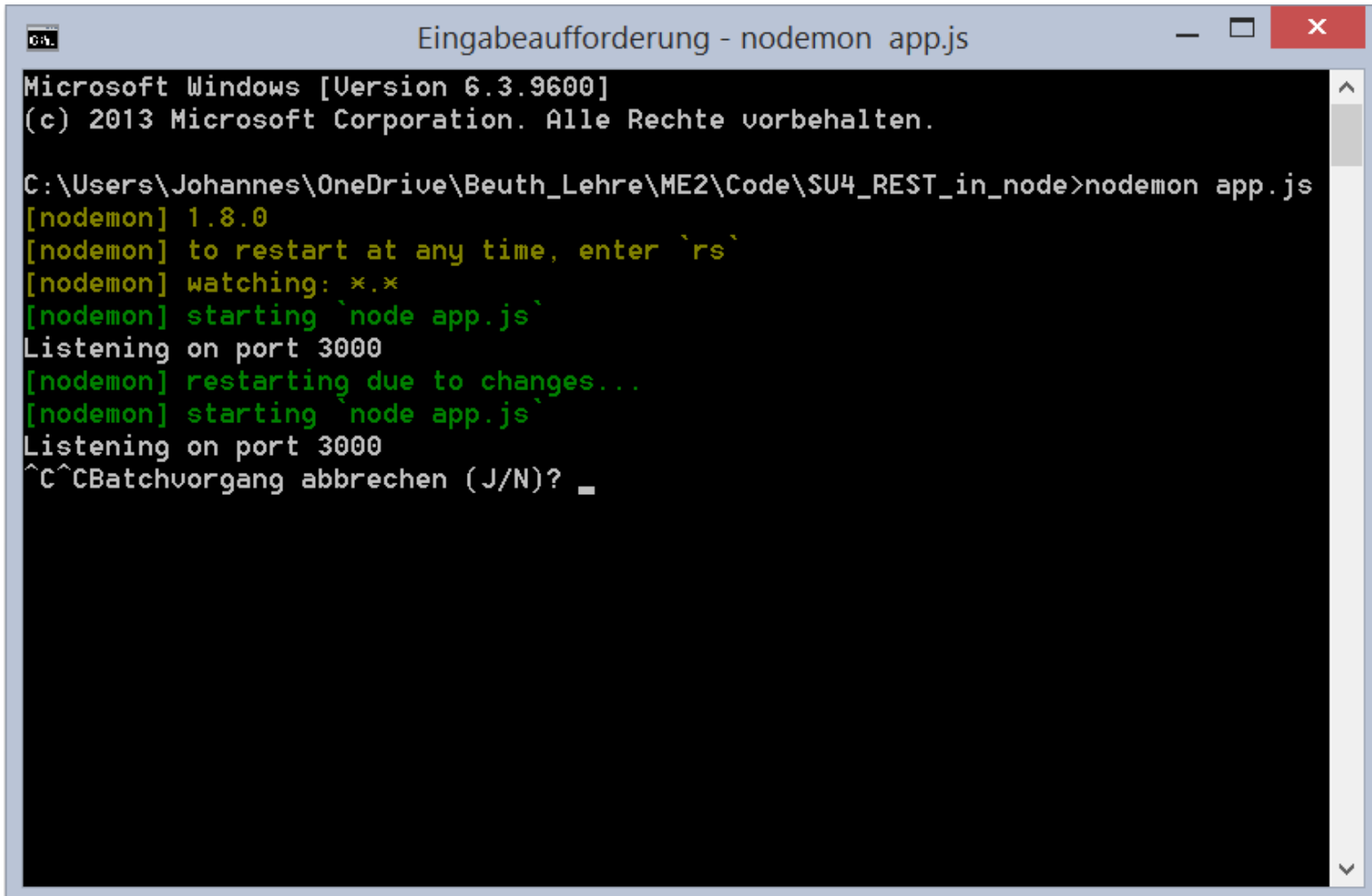


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU4_REST_in_node>nodemon app.js
[nodemon] 1.8.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Listening on port 3000
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Listening on port 3000
```

## Produktiv(er) entwickeln mit nodemon

- **nodemon app.js** (Beenden mit STRG-C\*\*)



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU4_REST_in_node>nodemon app.js
[nodemon] 1.8.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Listening on port 3000
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Listening on port 3000
^C^CBatchvorgang abbrechen (J/N)? _
```

# Agenda

- Wiederholung
- Update zum Semesterplan
- Ziel: TwitterApp REST-API
- Aufbau einer nodejs Anwendung
  - Requires
  - Middleware: Unterschied `.use()`, `.get()`
  - Request und Response
  - Parameter auslesen
  - Fehlerbehandlung
- Code-Übung zum Aufbau der API
- Kapselung von Routes mit `express.Router()`
- Postman im Einsatz
- Modul nodemon
- Zusammenfassende Fragen
- Ausblick

## Zusammenfassende Fragen (NEU: Antworten dann nächstes Mal)

- Für welche zwei unterschiedlichen Programmier-Ziele werden
  - `app.route(...)` und
  - `express.Router()`verwendet?

## Zusammenfassende Fragen (NEU: Antworten dann nächstes Mal)

- Wann bietet es sich an, **nodemon** statt **node** zu verwenden?

## Zusammenfassende Fragen (NEU: Antworten dann nächstes Mal)

- Was ist der Unterschied zwischen den Methoden
  - `app.use(...)` und
  - `app.all(...)`
  - ?

## Ausblick / Nächster Unterricht

### ■ Debuggen, Testen von Server-Code



www.phdcomics.com

**Vielen Dank  
und bis  
zum nächsten Mal**



## Kopie des ausgeteilte Beispielcodes, Abschnitte A-L

```
* Best start with GET http://localhost:3000/tweets to see the JSON for it
*
* @author Johannes Konert
* @licence CC BY-SA 4.0
*
*/
"use strict";
```

H

```
// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');

// own modules imports
var store = require('./blackbox/store.js');
```

F

```
// creating the server application
var app = express();
```

G

```
app.use(express.static(path.join(__dirname, 'public')));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
```

J

```
// logging
app.use(function(req, res, next) {
 console.log('Request of type ' + req.method + ' to URL ' + req.originalUrl);
 next();
});
```

L

~~// user has SEND the wrong type~~

## Kopie des ausgeteilte Beispielcodes, Abschnitte A-L

```
// API-Version control. We use HTTP Header field Accept-Version
app.use(function(req, res, next){
 // expect the Accept-Version header to be NOT set or being 1.0
 var versionWanted = req.get('Accept-Version');
 if (versionWanted !== undefined && versionWanted !== '1.0') {
 // 406 Accept-* header cannot be fulfilled.
 res.status(406).send('Accept-Version cannot be fulfilled').end();
 } else {
 next(); // all OK, call next handler
 }
});
```

A

```
// request type application/json check
app.use(function(req, res, next) {
 if (['POST', 'PUT'].indexOf(req.method) > -1 &&
 !(/application\/json/.test(req.get('Content-Type')))) {
 // send error code 415: unsupported media type
 // user has SEND the wrong type
 res.status(415).send('wrong Content-Type');
 } else if (!req.accepts('json')) {
 // send 406 that response will be application/json and
 // request does not support it by now as answer
 // user has REQUESTED the wrong type
 res.status(406).send('response of appl./json only supported');
 }
 else {
 next(); // let this request pass through as it is OK
 }
});
```

C

## Kopie des ausgeteilte Beispielcodes, Abschnitte A-L

```
app.get('/tweets', function(req,res,next) {
 res.json(store.select('tweets'));
});

app.post('/tweets', function(req,res,next) {
 // TODO check that the element is really a tweet!
 var id = store.insert('tweets', req.body);
 // set code 201 "created" and send the item back
 res.status(201).json(store.select('tweets', id));
});
```

K

```
app.get('/tweets/:id', function(req,res,next) {
 res.json(store.select('tweets', req.params.id));
 req.param('offset');
});

app.delete('/tweets/:id', function(req,res,next) {
 store.remove('tweets', req.params.id);
 res.status(200).end();
});

app.put('/tweets/:id', function(req,res,next) {
 store.replace('tweets', req.params.id, req.body);
 res.status(200).end();
});
```

B

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
 var err = new Error('Not Found');
 err.status = 404;
 next(err);
});
```

D

## Kopie des ausgeteilte Beispielcodes, Abschnitte A-L

```
// development error handler
// will print stacktrace as JSON response
app.use(function(err, req, res, next) {
 console.log('Internal Error: ', err.stack);
 res.status(err.status || 500);
 res.json({
 error: {
 message: err.message,
 error: err.stack
 }
 });
});
```

I

```
// Start server *****
app.listen(3000, function(err) {
 if (err !== undefined) {
 console.log('Error on startup, ',err);
 }
 else {
 console.log('Listening on port 3000');
 }
});
```

E