

Web Engineering II

05 REST APIs mit node.js/express

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Agenda

- **Wiederholung**
- **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

1. Was sind die **vier Merkmale** einer guten API?
2. Welche HTTP Methoden werden für welche **CRUD Operationen** verwendet?
3. Welche HTTP Methode ist niemals **idempotent**, welche kann idempotent sein?
4. Ab welchem **REST-Level** müssen die Ressourcen eine eindeutige URL haben und die Operationen über die HTTP-Methoden durchgeführt werden?
5. Welche **REST-Level** gibt es?
6. Was ist **HATEOS** und was hat es mit REST zu tun?
7. Welches REST-Level erfüllt die flickr-API, bei der wir mittels HTTP GET an <https://api.flickr.com/services/rest/?lat=..&lon=..&method=flickr.photos.search> angefragt haben?
8. Welche vier **HTTP Status-Code Gruppen** gibt es?
9. Wie unterstützen Sie REST-konforme (**zustandslose**) **Authentifizierung**?
10. Wie versionieren Sie eine REST-API?
11. Wozu ist die Unterstützung eines **?expand=** GET-Parameters gut?
12. Wie beschränken Sie die Menge an Einträgen beim Abruf von Ressourcen-Listen?
Welche Teile von HATEOS wären hier gut/sinnvoll?
13. Wie erfährt ein Client nach dem **HTTP POST** die gerade neu erstellte ID des Eintrages?
14. **Bonus-Frage:** Wie sehen die Ressourcen-URLs einer REST-Level 2 API aus für n:m Beziehungen bspw. von Person:Ort ("Angesehen")?

Wiederholung

Bonusfrage: Wie sehen die Ressourcen-URLs einer REST-Level 2 API aus für n:m Beziehungen bspw. von Person:Ort ("Angesehen")?

■ **`http://localhost:3000/api/visits/v4w3x1`**

■ **Datenschema bspw.:
(JSON Response)**

```
{  "id" : "v4w3x1",
   "personid" : "a1b2c3",
   "placeid" : "p1q3r5"
}
```

■ **oder auch:**

```
{  "href": "http://localhost:3000/visits/v4w3x1",
   "id" : "v4w3x1",

   "person" : {
     "href": "http://localhost:3000/person/a1b2c3"
   },

   "place" : {
     "href": "http://localhost:3000/place/p1q3r5"
   }
}
```

Agenda

- **Wiederholung**

- **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`**

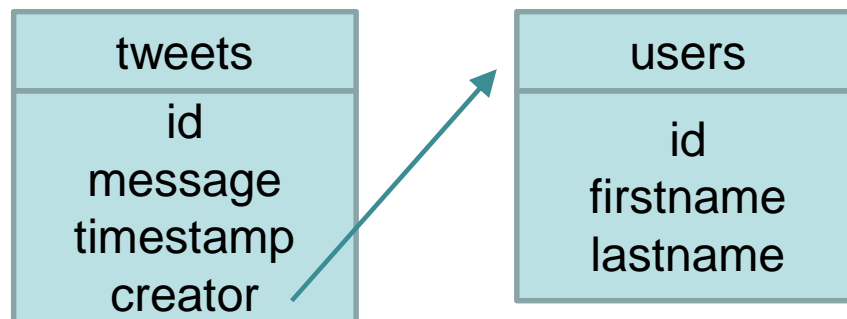
- **Zusammenfassende Fragen**

- **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

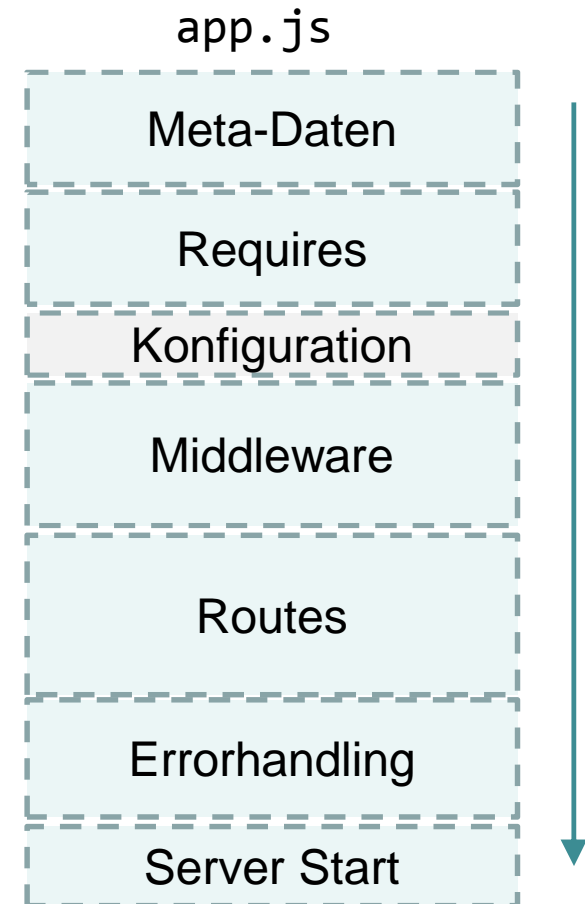
Agenda

- **Wiederholung**
- **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:

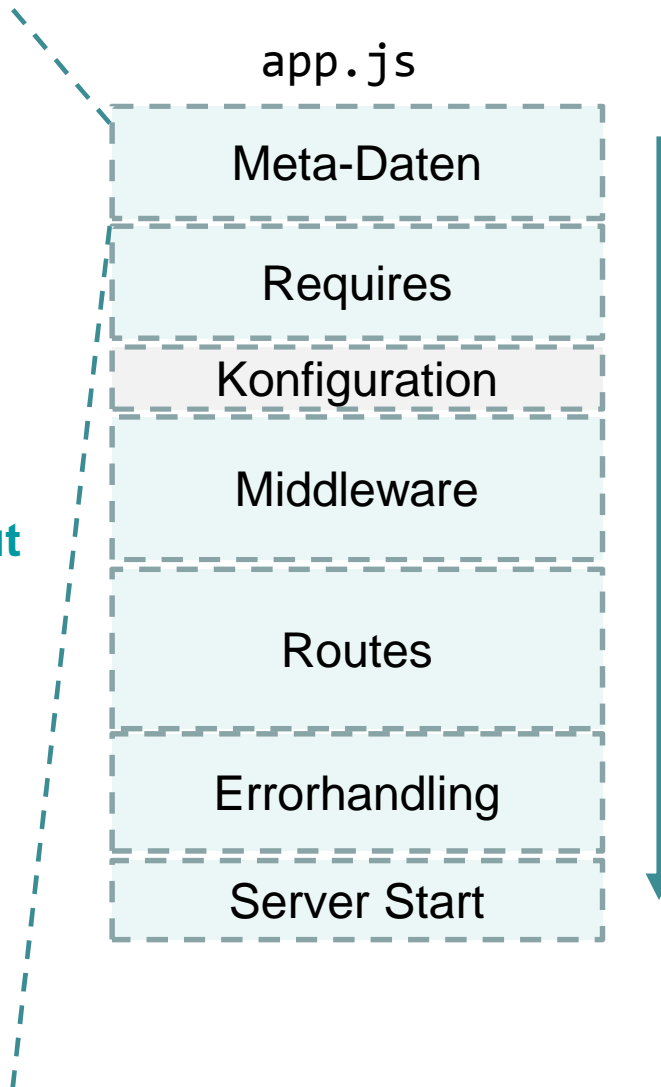
- von oben nach unten
- erst Handler registrieren
- dann Starten



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**
- **Ziel: TwitterApp REST-API**
- **Aufbau einer nodejs Anwendung**

Requires

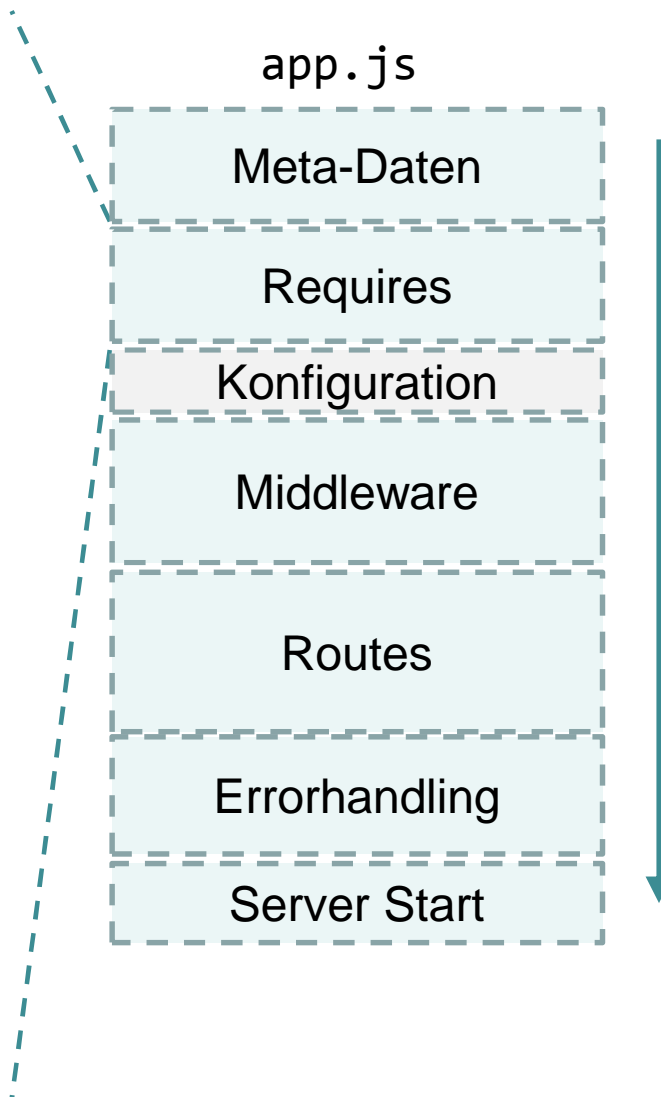
- **Middleware: Unterschied .use(), .get()**
- **Request und Response**
- **Parameter auslesen**
- **Fehlerbehandlung**
- **Code-Übung zum Aufbau der API**
- **Kapselung von Routes mit express.Router**
- **Postman im Einsatz**
- **Modul nodemon**
- **Zusammenfassende Fragen**
- **Ausblick**

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)

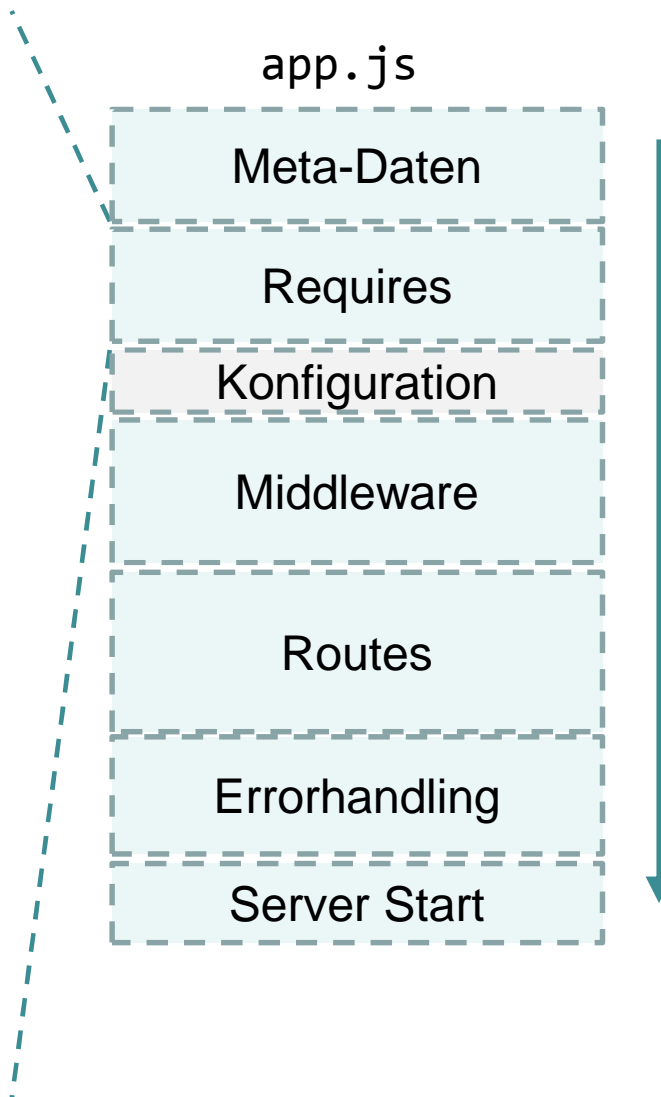
```
// node module imports
var path = require('path');
var express = require('express');
var bodyParser = require('body-parser');

// 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: in-Memory-Speicherung von Objekten,
damit Sie zunächst keine DB brauchen.

app.js

Meta-Daten

Requires

Konfiguration

Middleware

Routes

Errorhandling

Server Start

Agenda

- **Wiederholung**
- **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)`

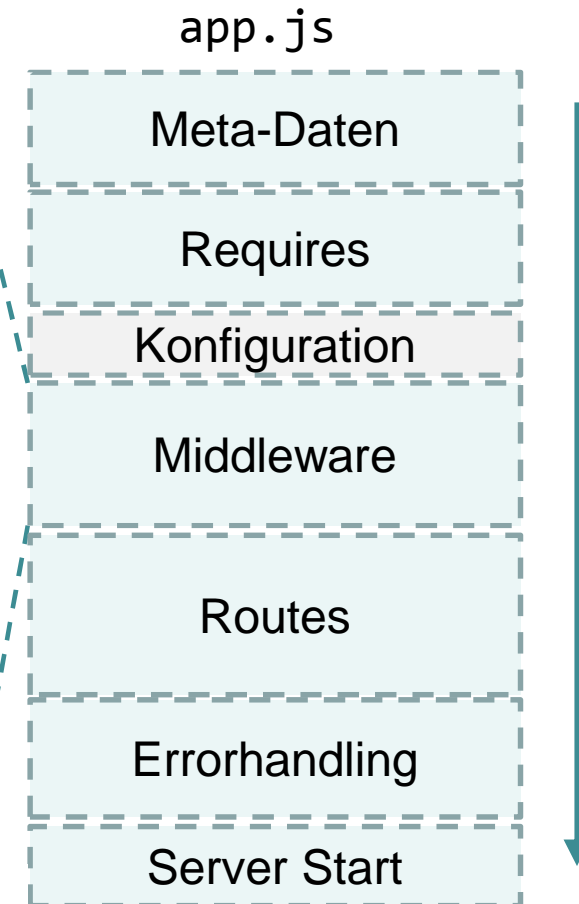
- trifft auf GET HTTP Methode zu
- pattern ist ein Muster (String oder RegExp)
- Handler ist Function

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

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

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

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

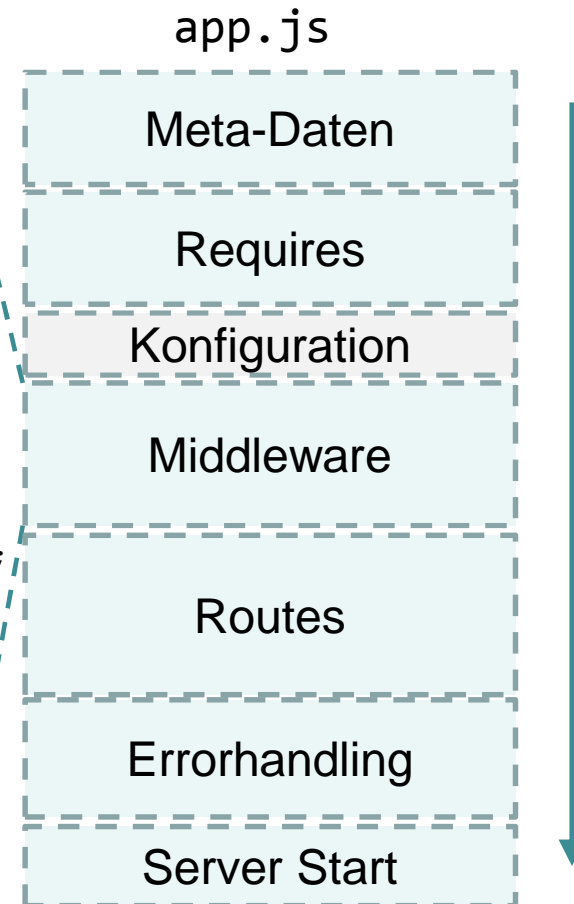


Node.js Aufbau einer Server-Anwendung (Details)

- 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**
- **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')
```



```
req.get('Content-Type')
```
- inkl. Convenience Methoden

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



```
req.is('json')
```

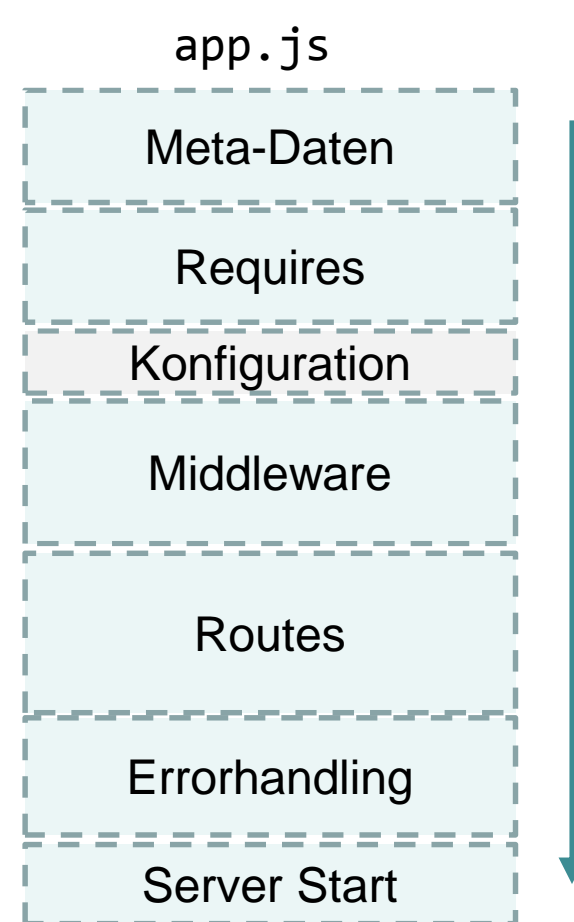
■ Response-Objekt ebenfalls

- ```
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 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`)

```
var val = req.query.key;
```

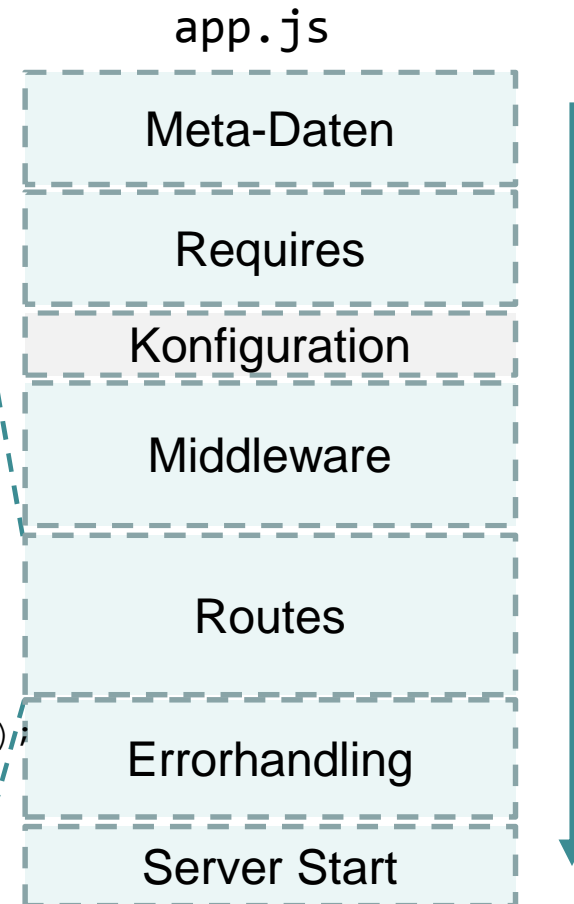
- POST-Parameter

(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**
- **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 **next mit Parameter** aufgerufen wird!

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

app.js

Meta-Daten

Requires

Konfiguration

Middleware

Routes

Errorhandling

Server Start

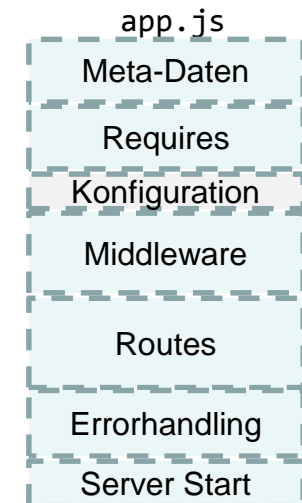
## Agenda

- **Wiederholung**
- **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:  
Welche Variante passt am Besten zu Ihrer Lösung



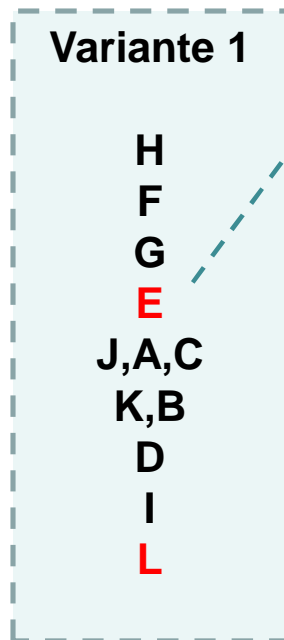
| Variante 1 | Variante 2 | Variante 3 | Variante 4 | Variante 5 |               |
|------------|------------|------------|------------|------------|---------------|
| H          | H          | H          | H          | Sonstige   | app.js        |
| F          | F          | F          | F          |            | 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          |            |               |

(Buchstaben in einer Zeile sind in der Reihenfolge beliebig)

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

# Node.js Aufbau einer Server-Anwendung

## Lösungen

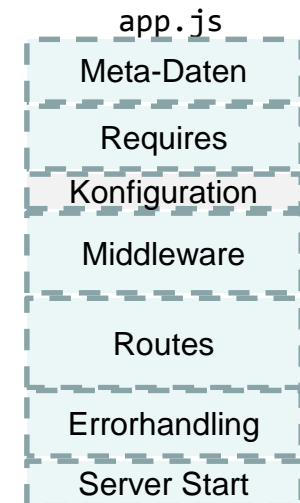


Server Start

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

### Problem:

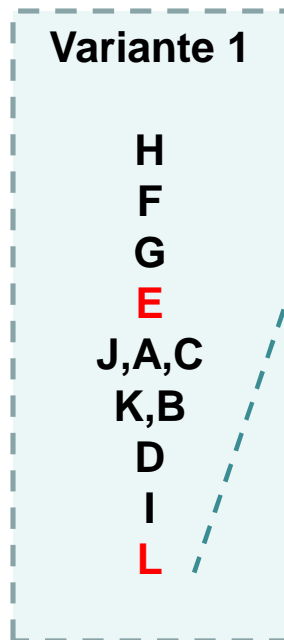
- Server nimmt bereits Anfragen entgegen
- Nicht alle Middleware und Routen sind registriert
- Potentielle Quelle für Fehler, Sicherheitslöcher und DB-Inkonsistenzen



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

# Node.js Aufbau einer Server-Anwendung

## Lösungen



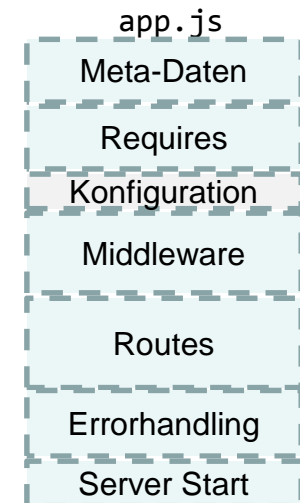
Logging  
Middleware

`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



### Variante 2

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

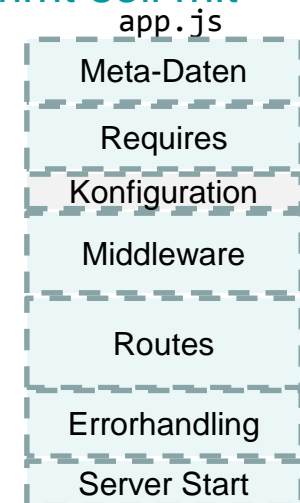
### Errorhandling

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

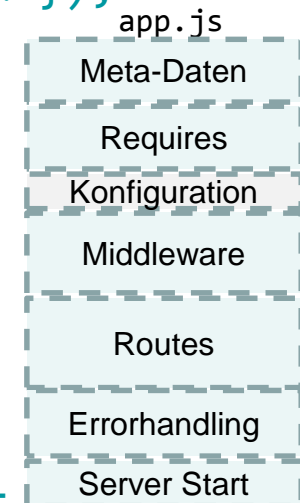
H  
F  
G  
J,L  
**D**  
I  
A,C  
K,B  
E

### 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.



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

# Node.js Aufbau einer Server-Anwendung

## Lösungen



### Variante 4

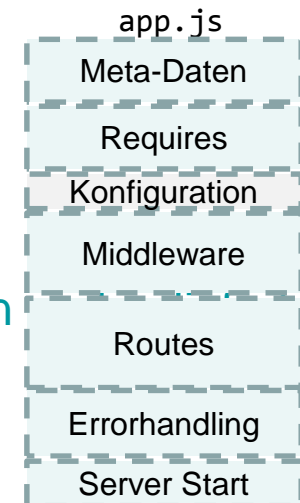
H  
F  
G  
J,A,C  
K,B  
D  
I  
L  
E

static files  
Middleware

Express.static Middleware zum Ausliefern statischer Dateien von der Festplatte

### Problem:

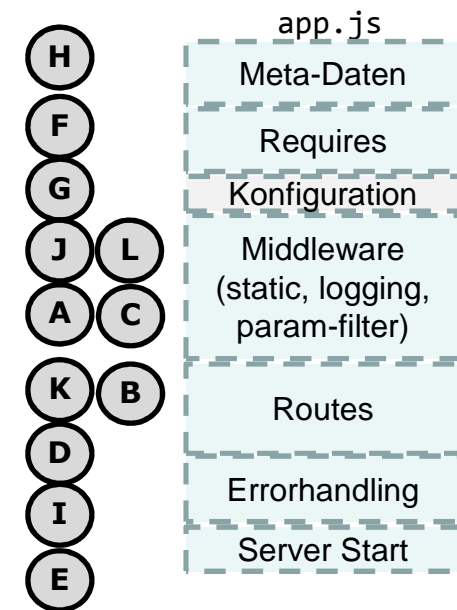
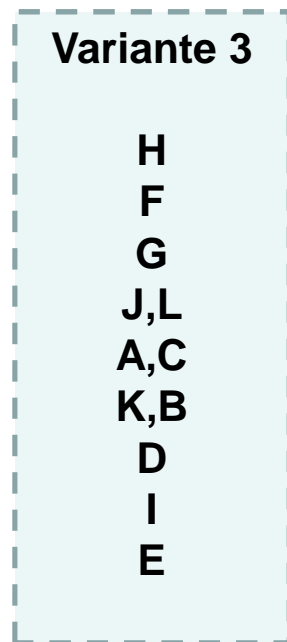
- Statische Dateien sollten vor Auswertung spezifischer Parameter der REST-API gefunden werden, sonst muss auch für bspw. index.html die Version und HTTP-Methode stimmen (A,C)



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

## ■ Lösungen



## Agenda

- **Wiederholung**
  - **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

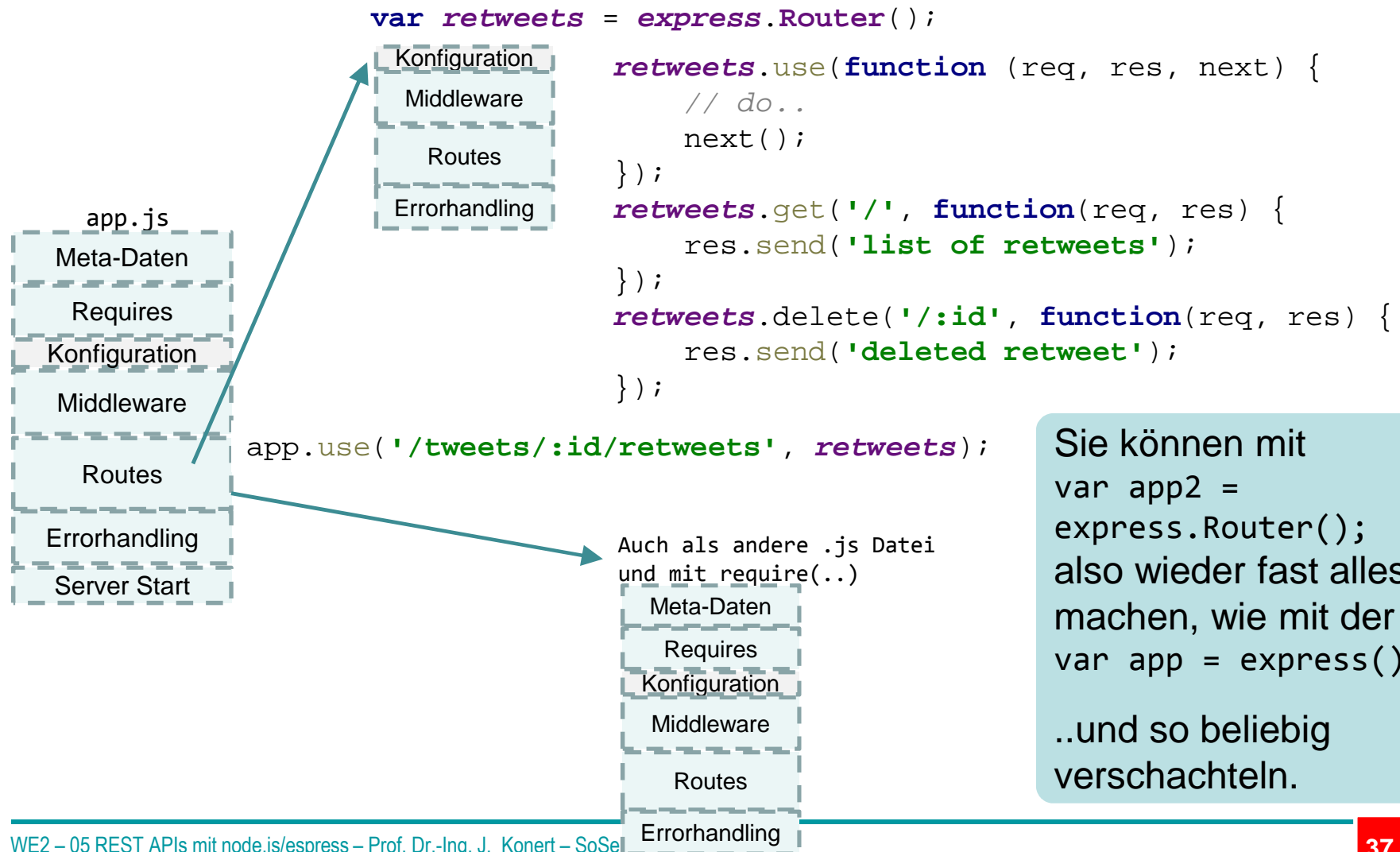
- **app.route()** erlaubt das Zusammenfassen verschiedener HTTP Methoden für gleiche Route

```
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

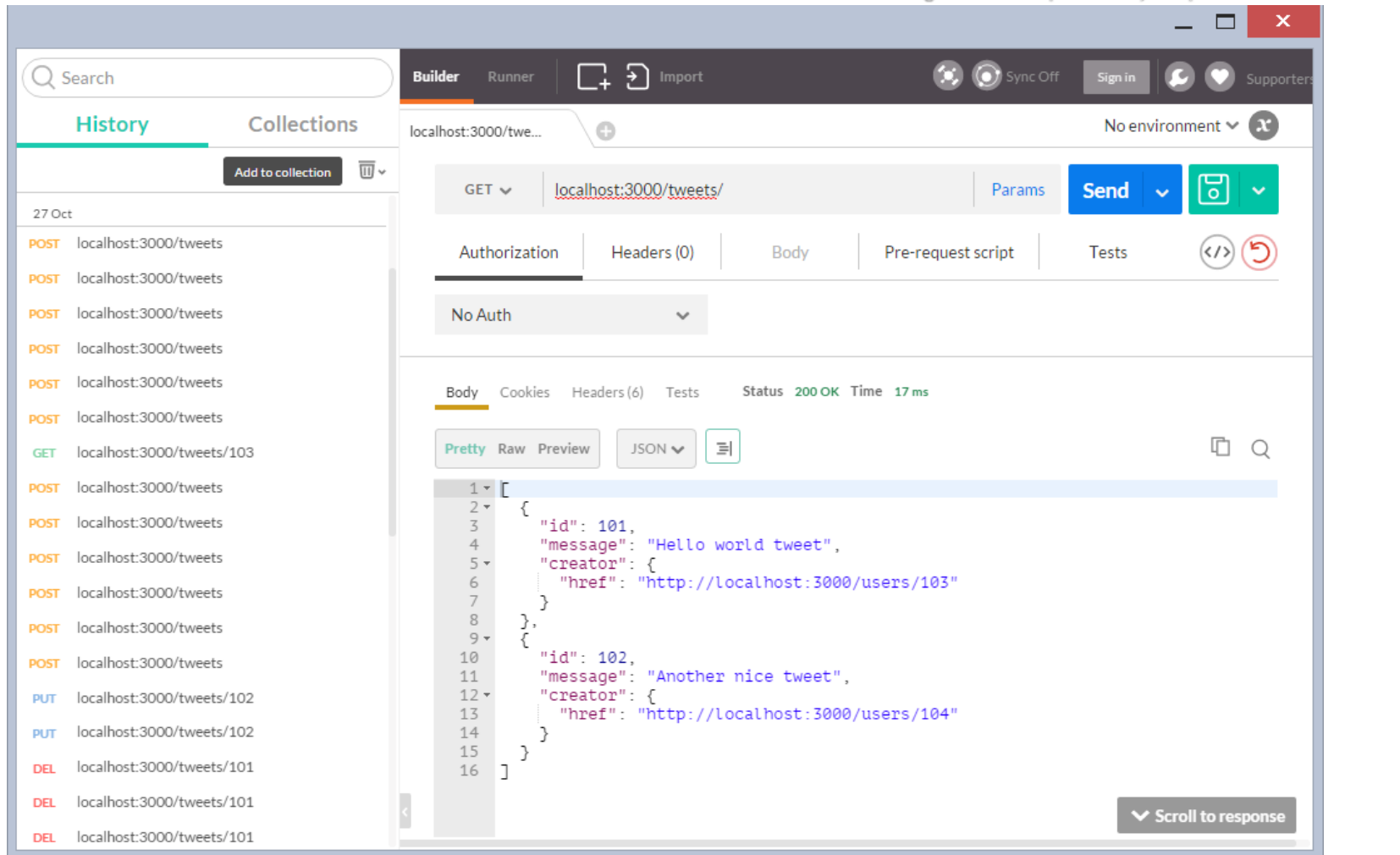


## Agenda

- **Wiederholung**
- **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 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/' with a status of '200 OK' and a response time of '17 ms'. The response body is displayed in JSON format, showing an array of two tweet objects:

```
[
 {
 "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**
- **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**

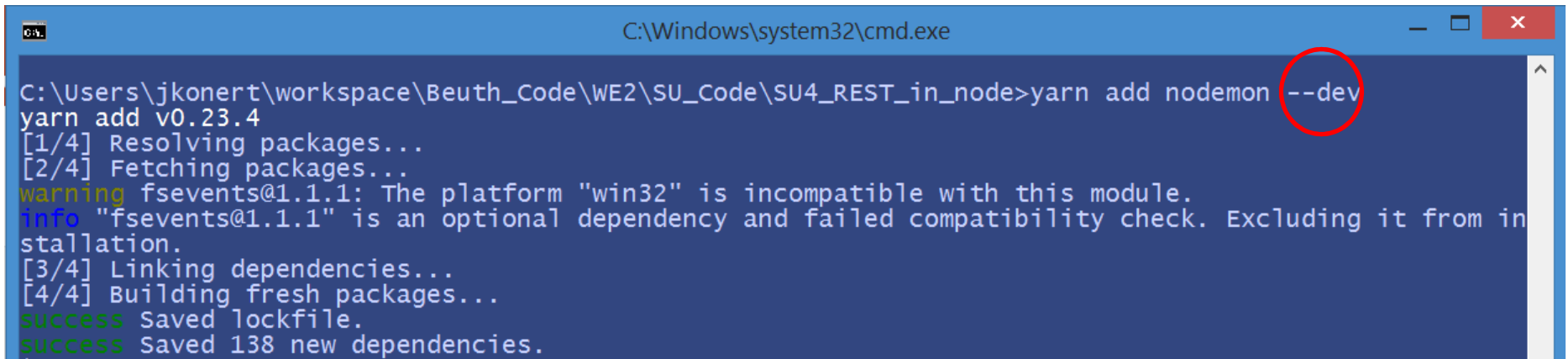
## Produktiv(er) entwickeln mit nodemon

### ■ Modul nodemon bietet

- Verwendung wie node zum Ausführen von bspw. app.js Code
- Überwacht alle geladenen Ressourcen
- Bei Änderung erfolgt automatischer Neustart

### ■ Installation?

- `yarn add nodemon --dev`
- oder `yarn global add nodemon`



```
C:\Windows\system32\cmd.exe

C:\Users\jkonert\workspace\Beuth_Code\WE2\SU_Code\SU4_REST_in_node>yarn add nodemon --dev
yarn add v0.23.4
[1/4] Resolving packages...
[2/4] Fetching packages...
warning fsevents@1.1.1: The platform "win32" is incompatible with this module.
info "fsevents@1.1.1" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 138 new dependencies.
```

## Produktiv(er) entwickeln mit nodemon

- **Paketmanager yarn erlaubt die Installation „global“, also ins nodejs Installations-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
  - `yarn global add <modulname>`
- **Paketmanager yarn erlaubt die Installation als „devDependencies“, also ins Projektverzeichnis, aber in der package.json nur für den development mode**
  - **Gut für:** Test-Module, Entwicklungshelfer, MiniWebServer, Code-Optimierer usw.
  - **Schlecht für:** Module, die auch auf dem Produktivserver benötigt werden;
  - `yarn add <modulname> --dev`

## Beispiel package.json

- **Global installierte Module sehen sie nicht, daher auch nicht bei Team-Mitgliedern installiert mit `yarn install` !**

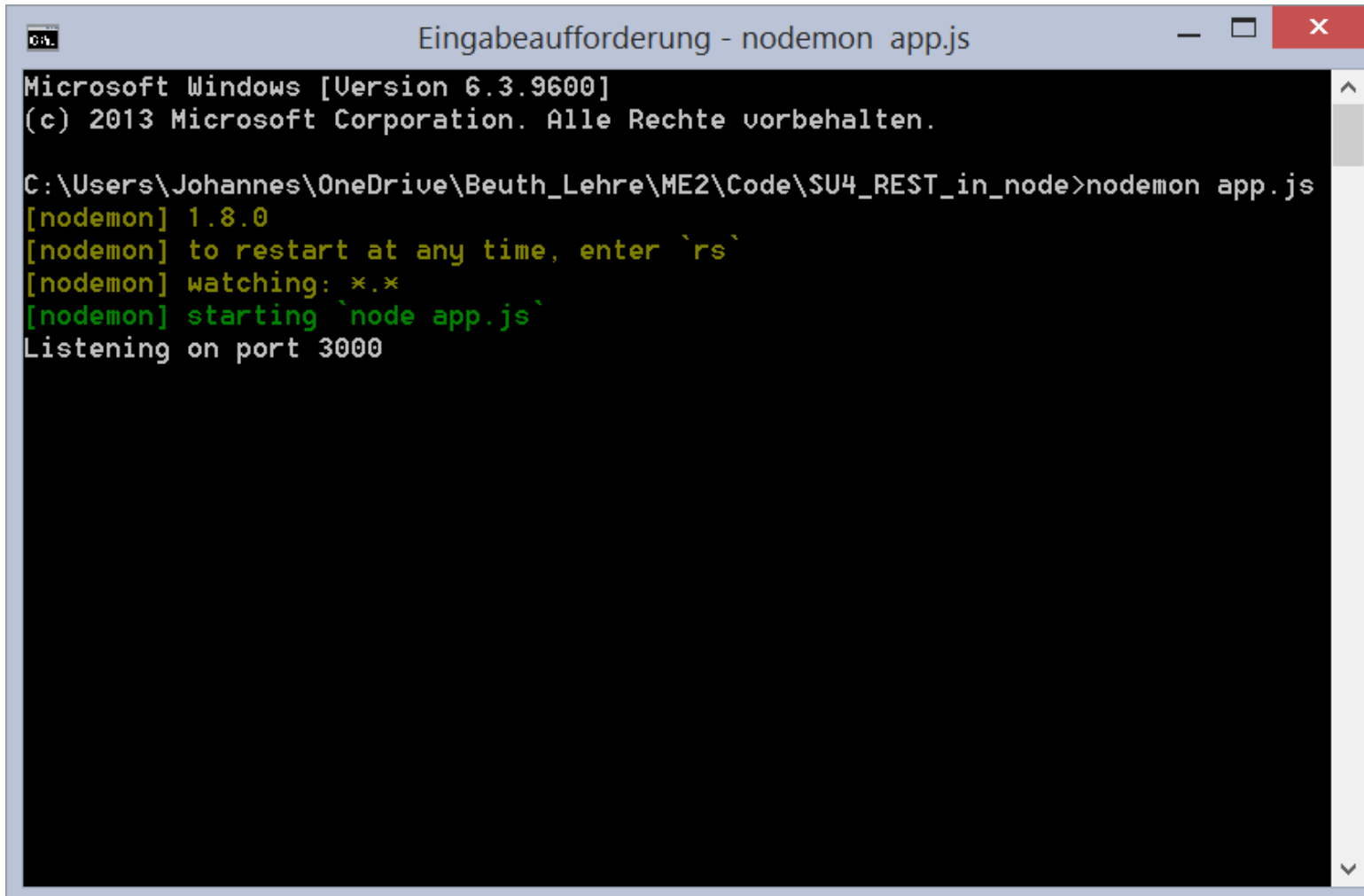
### Package.json

```
{
 "name": "we2-rest_in_node",
 "version": "0.0.1",
 "author": "Johannes Konert",
 "licence": "CC BY-SA 4.0",
 "private": true,
 "main": "app.js",
 "scripts": {
 "start": "node app.js"
 },
 "dependencies": {
 "body-parser": "^1.17.1",
 "debug": "^2.6.6",
 "express": "^4.15.2"
 },
 "devDependencies": {
 "nodemon": "^1.11.0"
 }
}
```



## Produktiv(er) entwickeln mit nodemon

- **nodemon app.js**

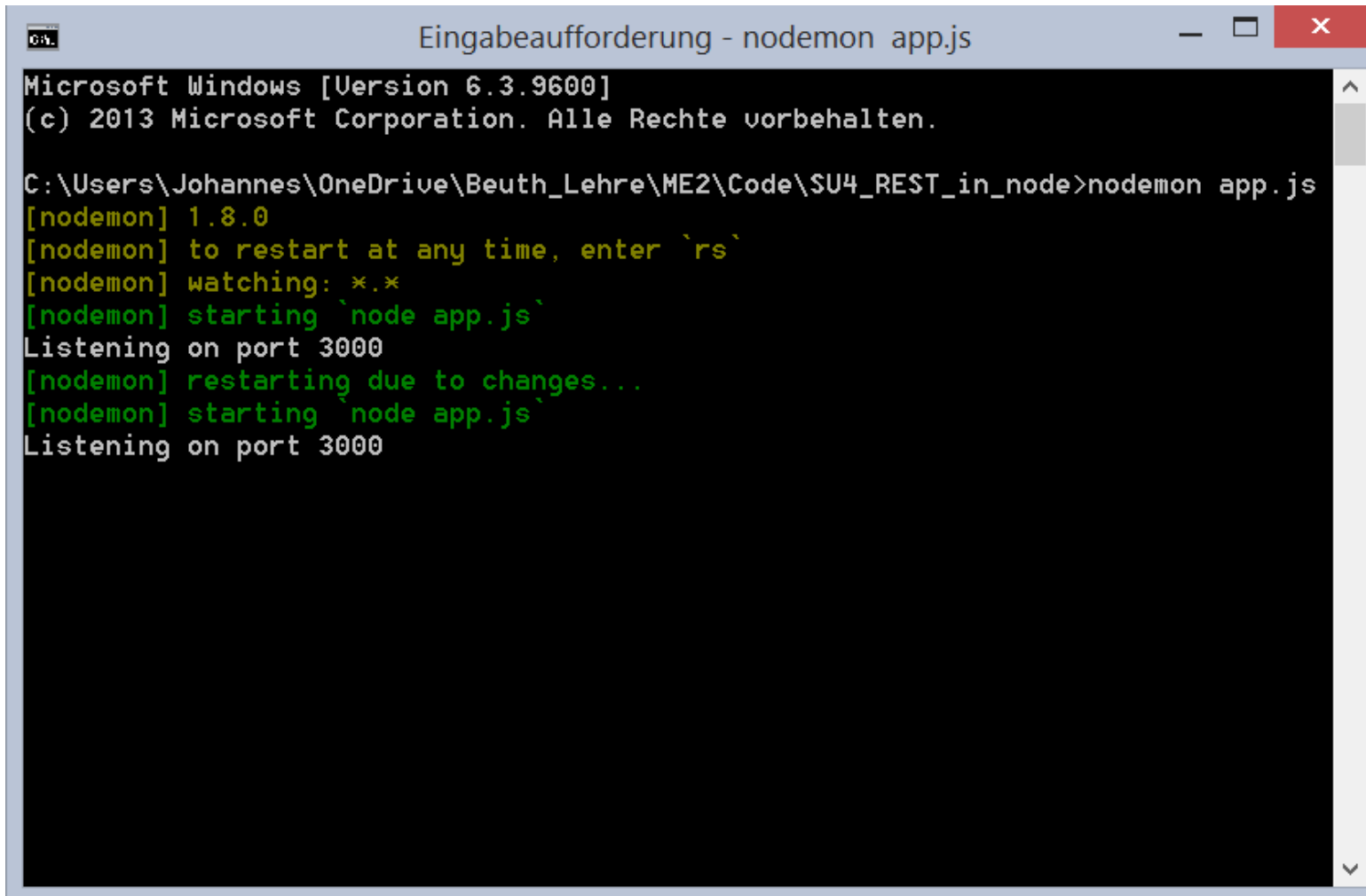


```
Eingabeaufforderung - nodemon app.js
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

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)

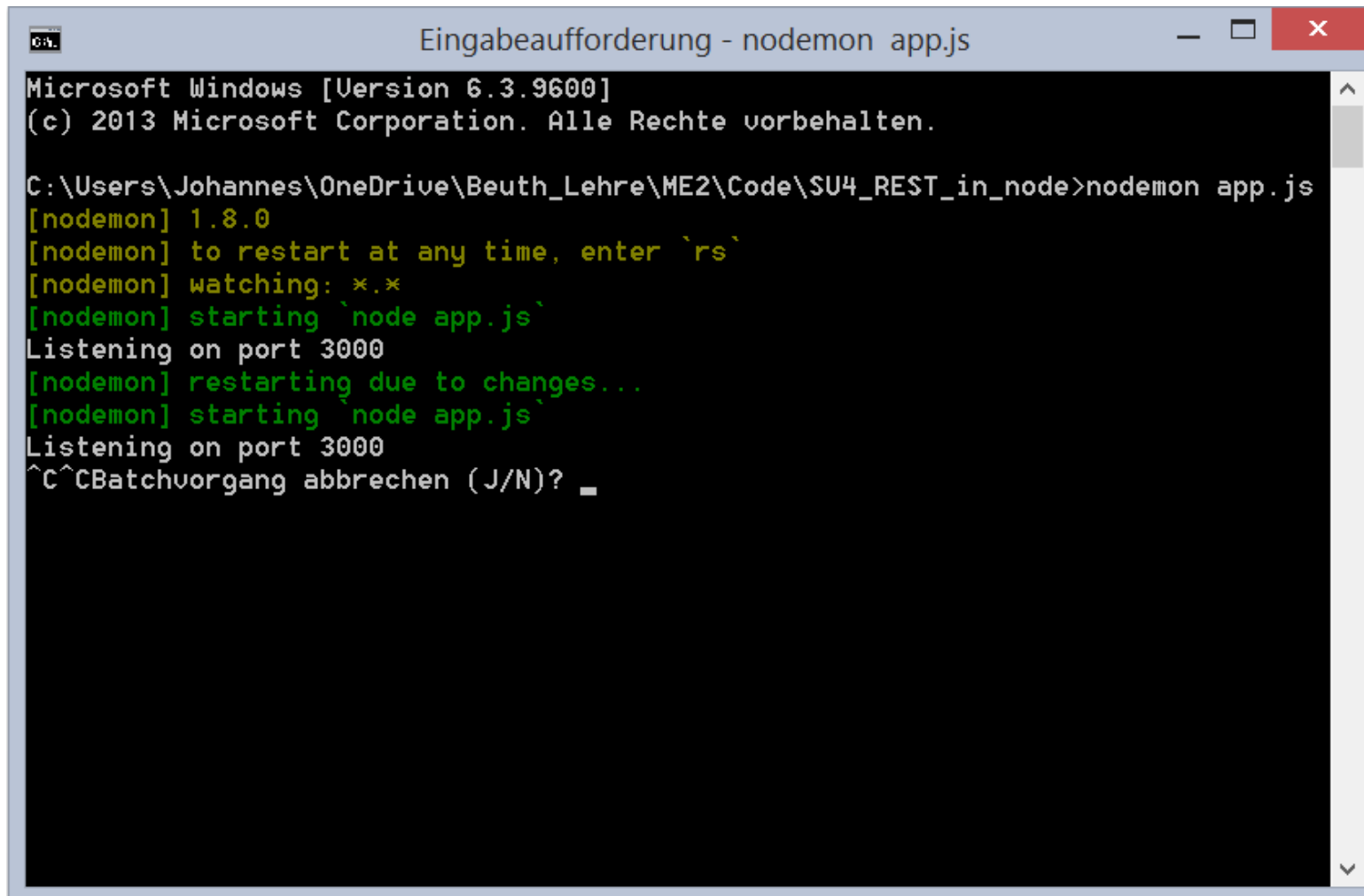


```
Eingabeaufforderung - nodemon app.js
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

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
[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\*\*)



```
Eingabeaufforderung - 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
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Listening on port 3000
^C^C Batchvorgang abbrechen (J/N)? _
```

## Agenda

- **Wiederholung**
- **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

1. Für welche zwei unterschiedlichen Programmier-Ziele werden `app.route(...)` und `express.Router()` verwendet?
2. Wann bietet es sich an `nodemon` statt `node` zu verwenden?
3. Was ist der Unterschied zwischen den Methoden `app.use(...)` und `app.all(...)`?
4. Wie können Teile der URL im `pattern` bei `app.all(,pattern', function handler)` als Variable deklariert werden, um diese im Handler-Code dann zu nutzen als Variable im Code genutzt werden (in `request.params`)?
5. Welche Unterschiede bestehen zwischen `request.params` und `request.query`?
6. Wie wandeln Sie den HTTP Body, der bei einem POST an Ihren Server geschickt wird, wieder von JSON String nach JavaScript Object um?
7. Warum ist es wichtig, dass der Server-Start mittels `app.listen(..)` als letzter Aufruf in Ihrer `app.js` erfolgt?
8. Welche Module installieren Sie global, welche als devDependencies und welche als normale dependencies in Ihrem Projekt? (bspw. `nodemon`, `bodyparser` und `node-minify`)
9. Wozu dient der Aufruf von `next()` in einer Handler-Funktion? Welcher Unterschied besteht zwischen `next()` und `next(obj)` ?
10. Wozu dient `"use strict";` in einer JavaScript-Datei?
11. Warum ist es essentiell, dass Sie im Server auf jeden (auch ungültige) Request eine Antwort senden? Wie machen Sie das? Welchen Statuscode verwenden Sie bei ungültigen Anfragen?
12. **Bonusfrage:** Wie können Sie mehrere Handlerfunktionen in express für die gleiche Route-URL angeben (die sich dann nacheinander mit `next()`) aufrufen? Wie können Sie umgekehrt für mehrere Routes die gleiche Handlerfunktion angeben? (in beiden Fällen ohne copy&paste)

## Ausblick / Nächster Unterricht

- Debuggen, Testen von Server-Code



**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