

Multimedia Engineering II

07 Vertiefung, Wiederholung

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Ihre Rückmeldungen im Etherpad

Wiederholung

- Closures **x**
- Warten auf asynchrone Funktionen (Callbacks)
- Postman detaillierter

Vertiefung

- Wie nutzt man mySQL oder postgresql mit node? **xxxx**
- Wie macht man node.js Server sicher(er)? **xxxx**
- Skalierung von Node Applikationen **x**
- Promises sinnvoll einsetzen **x**
siehe u.a.
<https://gist.github.com/domenic/3889970>
- OAuth **x**
siehe u.a. Unterricht 10 MME2 (13.06.)
- Einführung in Ember.js

Agenda

- **Wiederholung zu Modularisierung**
- **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
- **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Zusammenfassende Fragen und Wiederholung

Wiederholung als Teamquiz

Der Raum wird in zwei Teams eingeteilt



1. Ein Teammitglied (A) zieht 2 Karten
 - Wählt eine von beiden aus
2. Stellt die Frage dem anderen Team (B)
 - Das Team darf diskutieren über die Lösung
3. Das Team (B) antwortet
 - Team A ergänzt.
4. Dozent als Schiedsrichter vergibt endgültigen Punkt an Team A oder B



Anschließend ist das andere Team dran (1.-3.)

Insgesamt werden **vier Fragen** besprochen.

Agenda

- **Wiederholung zu Modularisierung**
- **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
- **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Closure

Ein Closure ist eine Funktion,
die Elemente
Ihres umgebenden Definitionskontextes verwendet,
(obwohl dieser bereits beendet wurde).

```
var i = 100;  
var globalCounter = function(value) {  
    return ++value;  
};  
i = globalCounter(i); // 101  
// i = i+1;
```

kein Closure

```
var getCounter = function(value) {  
    var i = value;  
    return function() {  
        return ++i;  
    }  
};  
  
var globalCounter = getCounter(100);  
globalCounter() // 101
```

mit Closure

Closure

```
var getCounter = function(startValue) {  
    var i = startValue;  
    return function() {  
        return ++i;  
    }  
};  
  
var globalCounter = getCounter(100);  
globalCounter()
```

- **Anwendung von Closures immer dann,**
 - wenn eine Funktion auf bestimmte Elemente zugreifen soll, die nicht als Parameter übergeben werden
 - wenn Variablen oder Hilfsfunktionen vor äußerem Zugriff „versteckt“ werden sollen

Quellen/Texte zu Closures in der Praxis

- **Closures Herleitung über Definitionskontext**
<https://developer.mozilla.org/de/docs/Web/JavaScript/Closures>
- **Demystifying Closures**
<http://www.sitepoint.com/demystifying-javascript-closures-callbacks-iifes/>
- **Why use Closures? (and when)**
<http://howtonode.org/why-use-closure>


Agenda

- **Wiederholung zu Modularisierung**
- **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
- **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Warten auf asynchrone Funktionen mit Callbacks


Beispiel:

```
function handle() {  
    console.log(message);  
}  
function setAlarm(message, timeout) {  
    setTimeout(handle, timeout);  
}  
setAlarm("Wake UP!", 100);
```




ReferenceError:
message is not defined

```
function setAlarm(message, timeout) {  
    setTimeout(function() { console.log(message); }, timeout);  
}  
setAlarm("Wake UP!", 100);
```



```
function setAlarm(message, timeout) {  
    function handle() {  
        console.log(message);  
    }  
    setTimeout(handle, timeout);  
}  
setAlarm("Wake UP!", 100);
```



Warten auf asynchrone Funktionen mit Callbacks

Beispiel:

```
function setAlarm(message, timeout) {  
  function handle() {  
    console.log(message);  
  }  
  setTimeout(handle, timeout);  
}  
setAlarm("Wake UP!", 100);
```



Fazit:

Um auf eine asynchrone Funktion „warten“ zu können, muss diese async. Funktion eine **Callback-Funktion als Parameter** unterstützen, die erst dann ausgeführt wird, wenn die asynchrone Operation beendet ist.

Fazit:

Callback-Funktionen
sind ~immer~ auch Closure-fähig.

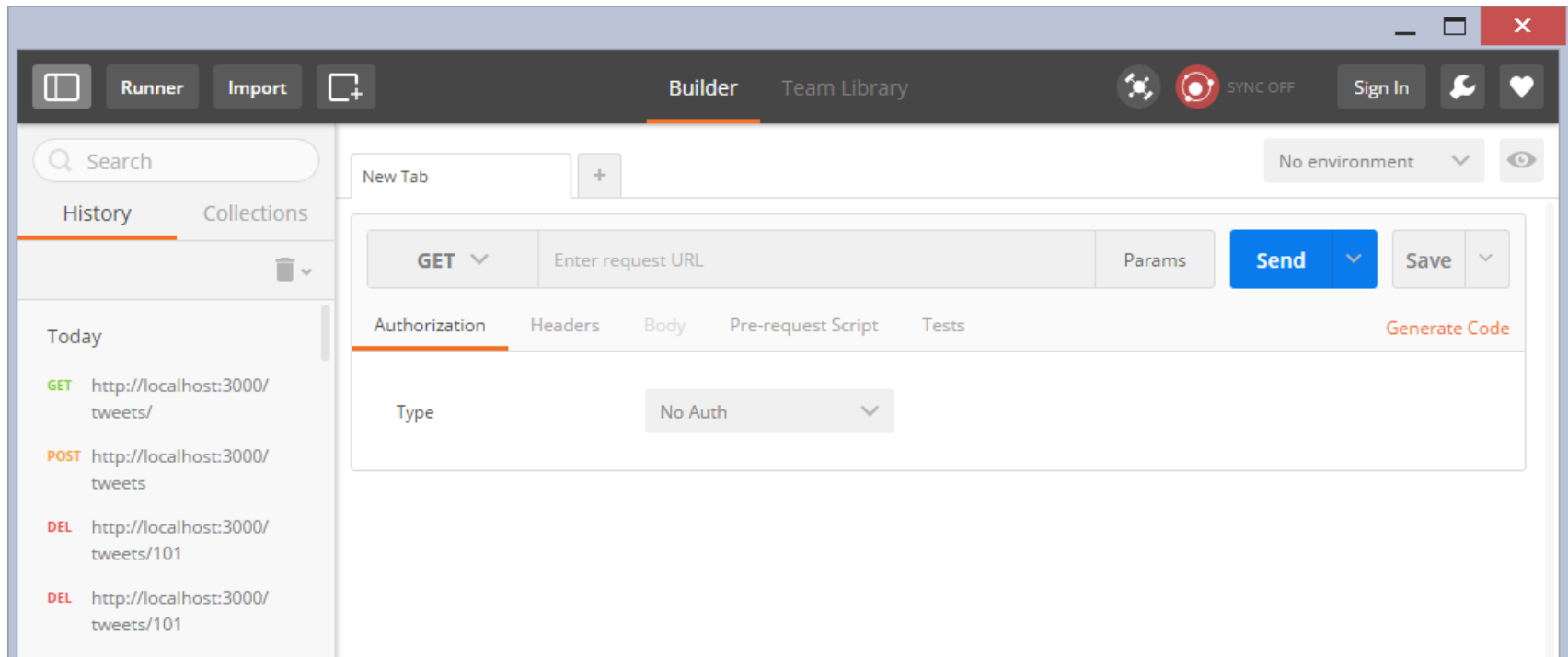
Agenda

- **Wiederholung zu Modularisierung**
- **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
- **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Postman im Detail

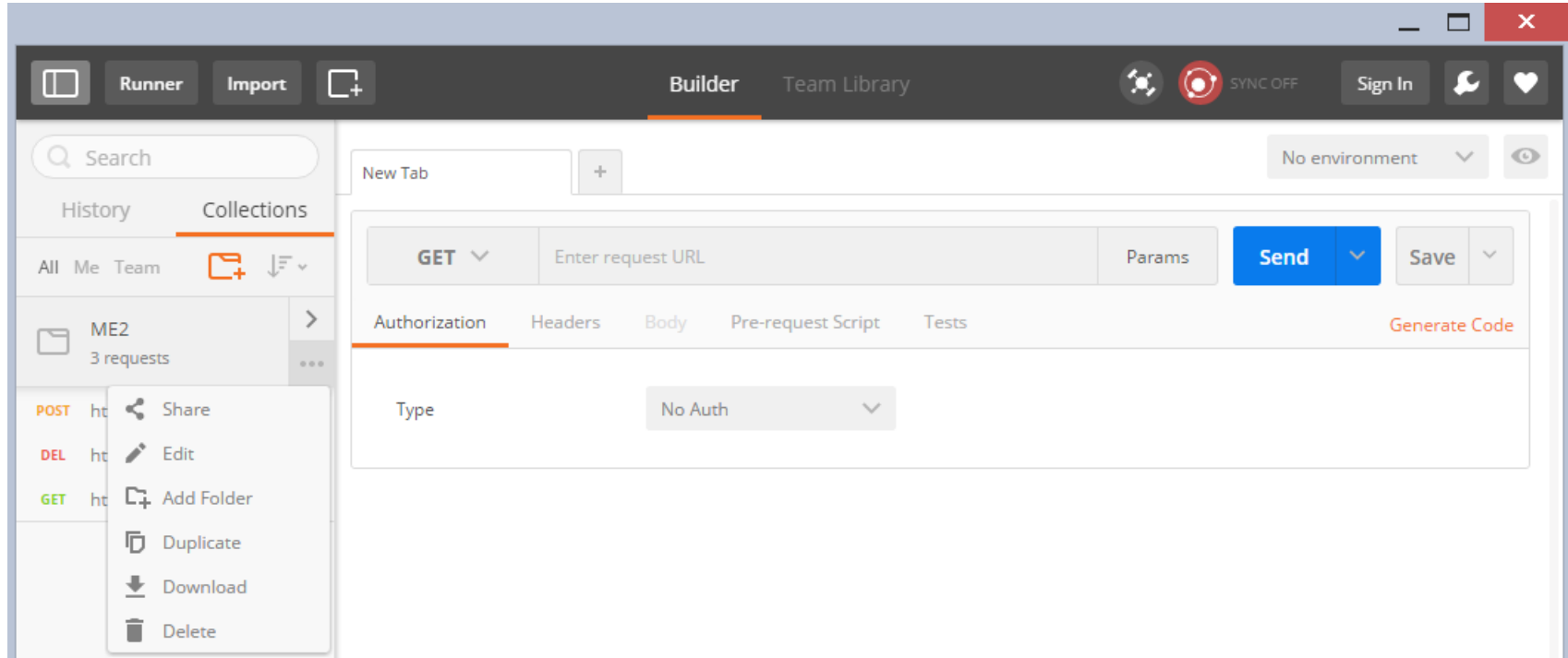
- **REST-Schnittstellen-Werkzeug**
- **Kostenlos für Google Chrome (Plugin)**
- **Unterstützt**
 - HTTP Methoden Anfragen (Header, Body)
 - Zusammenfassung von Requests zu Collections
 - Test-Abfragen bei Requests
 - Teamarbeit
 - Umgebungsdefinitionen

Postman im Detail



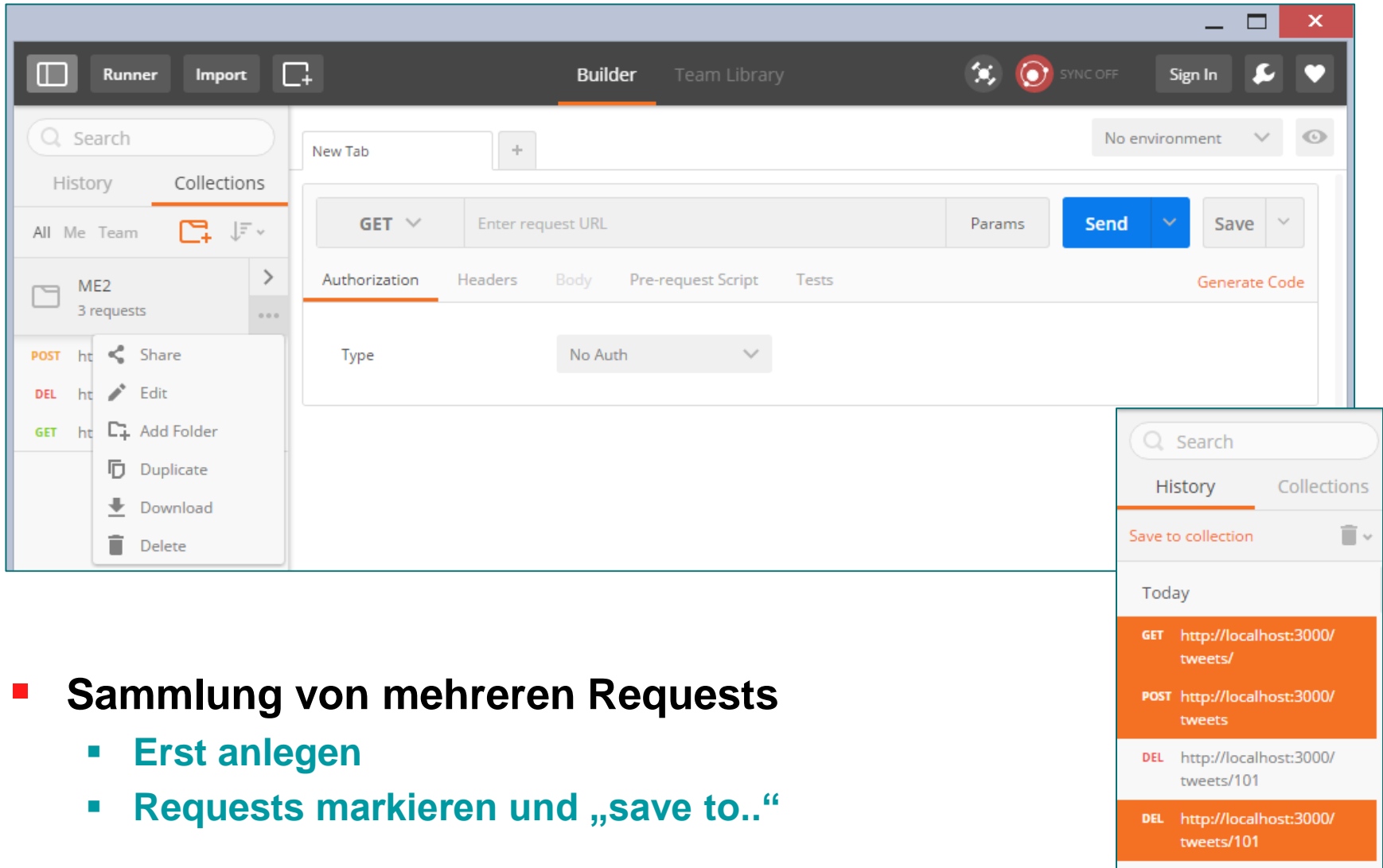
- **HTTP Methoden**
- **Achtung beim Scrollen im Response-Body:** Die eigenen Request-Header und Body-Zeilen scrollen weg (wieder sichtbar beim Hochscrollen!)

Postman im Detail: Collections



■ Sammlung von mehreren Requests

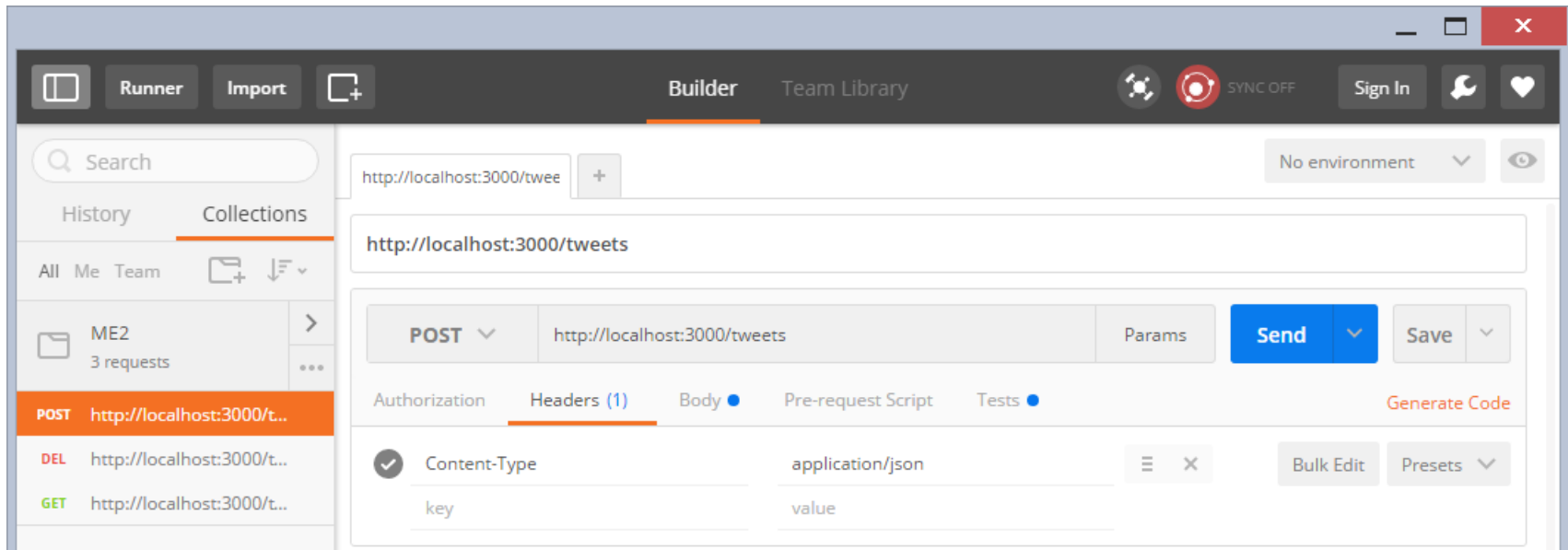
Postman im Detail: Collections



The screenshot displays the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'ME2' containing 3 requests. A context menu is open over the collection, listing actions: Share, Edit, Add Folder, Duplicate, Download, and Delete. The main workspace shows a 'GET' request with a 'No Auth' authorization type. The right sidebar shows a 'Save to collection' button and a list of requests under the 'Today' section.

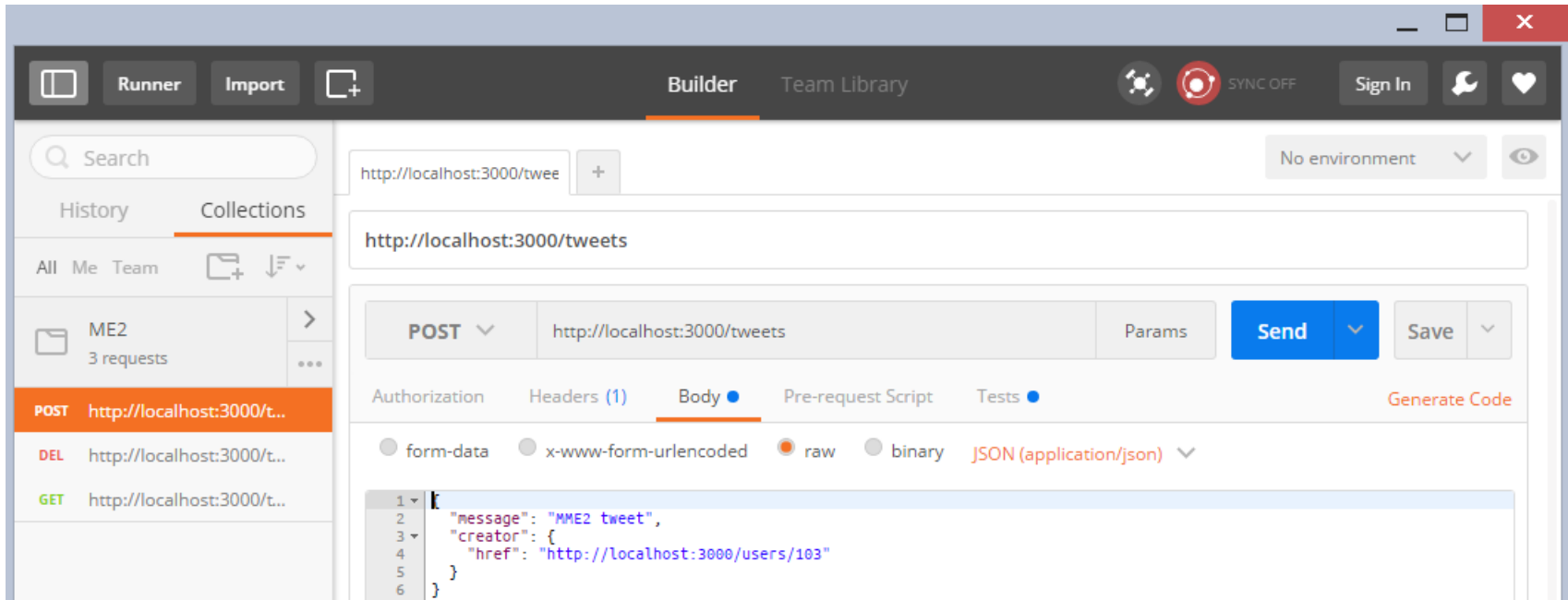
- **Sammlung von mehreren Requests**
 - Erst anlegen
 - Requests markieren und „save to..“

Postman im Detail: Requests absetzen



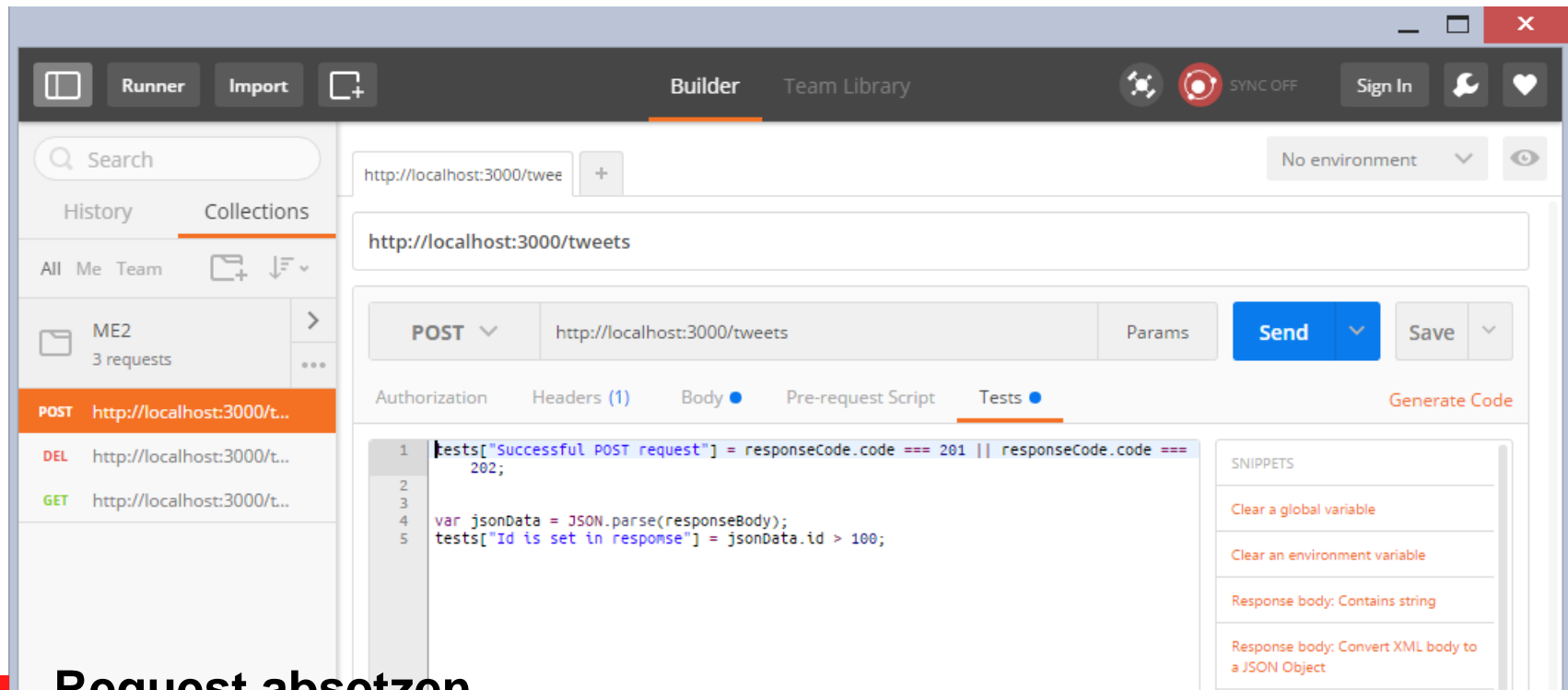
- **Request absetzen**
 - **URL angeben**
 - **Methode auswählen**
 - **Header-Felder setzen**
 - **Body-Inhalt setzen**
 - **Ggf. Tests spezifizieren (wenn man will)**

Postman im Detail: Requests absetzen



- **Request absetzen**
 - URL angeben
 - Methode auswählen
 - Header-Felder setzen
 - Body-Inhalt setzen
 - Ggf. Tests spezifizieren (wenn man will)

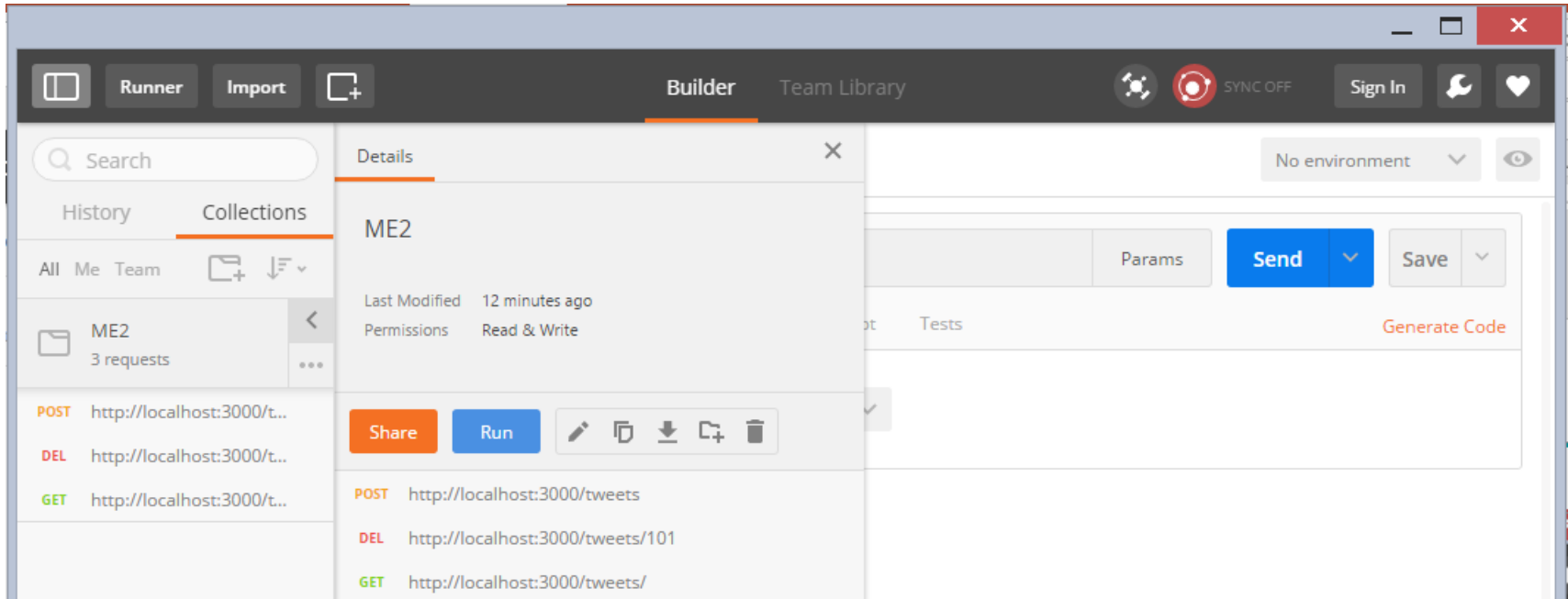
Postman im Detail: Requests absetzen



Request absetzen

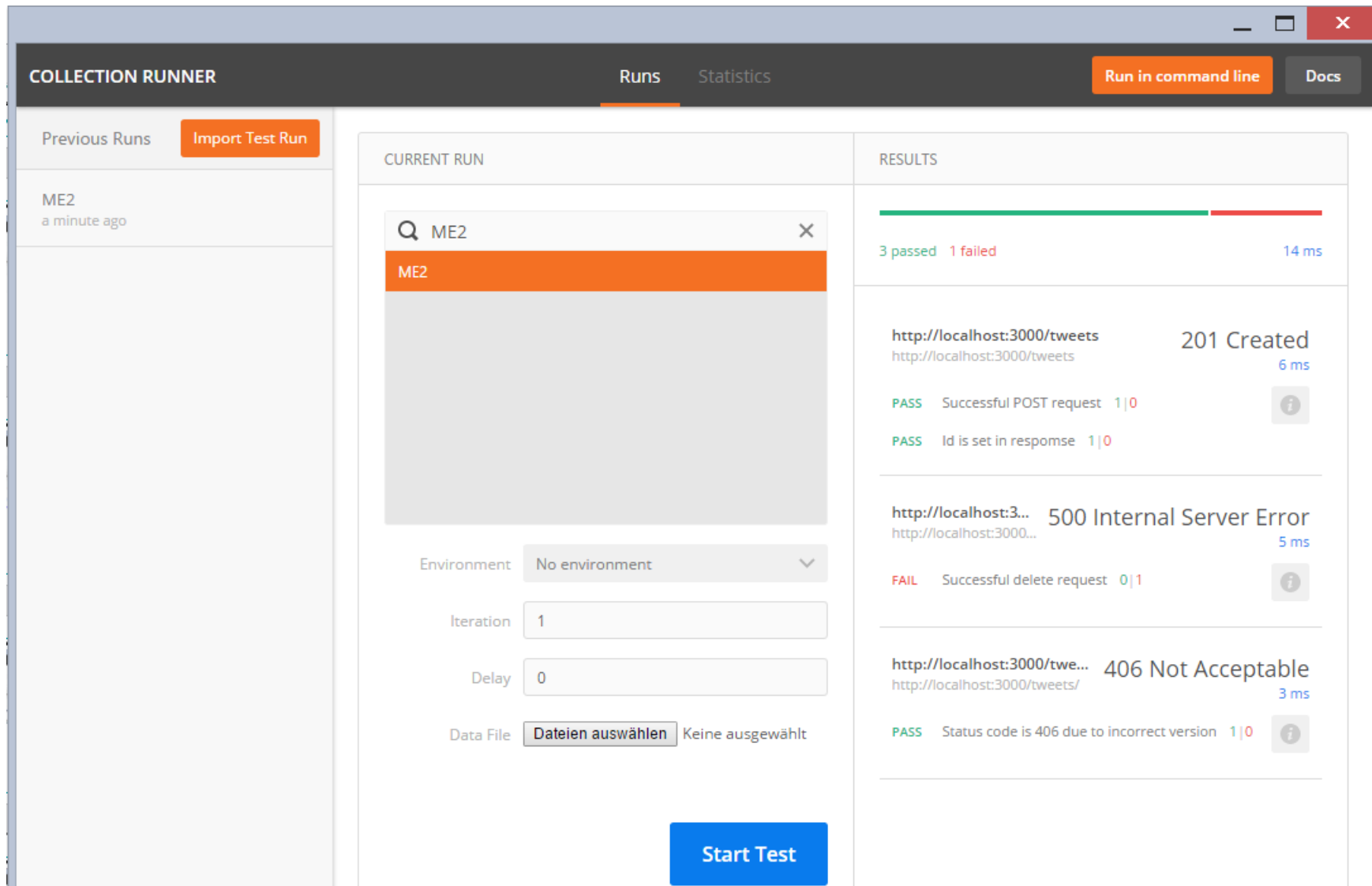
- URL angeben
- Methode auswählen
- Header-Felder setzen
- Body-Inhalt setzen
- Ggf. Tests spezifizieren (wenn man will)

Postman im Detail: Collections als Testsuite durchlaufen



- Collections, <, Run

Postman im Detail: Collections als Testsuite durchlaufen (Runner)



The screenshot displays the Postman Collection Runner interface. The top bar includes the 'COLLECTION RUNNER' title, tabs for 'Runs' and 'Statistics', and buttons for 'Run in command line' and 'Docs'. The left sidebar shows 'Previous Runs' with a list containing 'ME2' (a minute ago) and an 'Import Test Run' button. The main area is divided into 'CURRENT RUN' and 'RESULTS'.

CURRENT RUN

- Search bar: ME2
- Environment: No environment
- Iteration: 1
- Delay: 0
- Data File: Dateien auswählen (Keine ausgewählt)
- Start Test button

RESULTS

3 passed 1 failed 14 ms

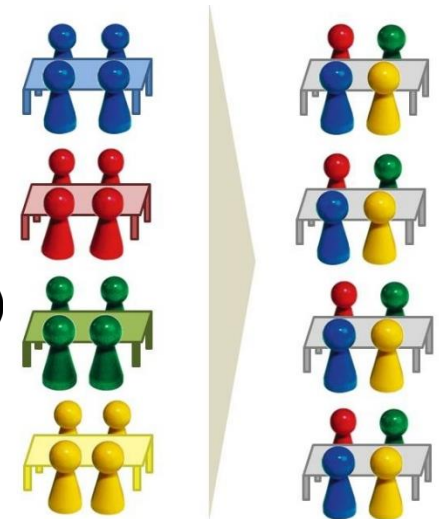
Request	Response	Status	Details
http://localhost:3000/tweets	201 Created	PASS	Successful POST request 1 0
http://localhost:3000/tweets		PASS	Id is set in response 1 0
http://localhost:3000/tweets	500 Internal Server Error	FAIL	Successful delete request 0 1
http://localhost:3000/tweets	406 Not Acceptable	PASS	Status code is 406 due to incorrect version 1 0

Agenda

- **Wiederholung zu Modularisierung**
- **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
- **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Expertengruppen: Ablauf

- **Phase 1: Bildung von Arbeitsgruppen zu den Themen (2min)**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Phase 2: Arbeitsgruppen werden zu Expertengruppen durch Textarbeit (10min), Recherche (5min) und Diskussion (10min)**
 - Was ist das?
 - Wie wendet man es an?
 - Wann?
 - Welche Grenzen/Nachteile gibt es?
 - ..
- **Phase3: Bildung neuer (Transfer)Gruppen, in welchen jeweils mind. ein Experte pro Thema sitzt (3x7=21min)**
 1. Nacheinander erläutern der wichtigsten Ergebnisse zu den W-Fragen
 2. Fragen aus der Runde klären
 3. Offene Fragen fürs Plenum notieren
- **Phase 4: Fragen im Plenum stellen und klären (10min)**



Quellen/Texte zum Verwenden von SQL in node.js

- MySQL <https://www.sitepoint.com/using-node-mysql-javascript-client/>
- PostgreSQL <http://mherman.org/blog/2015/02/12/postgresql-and-nodejs/#.V0MnguQl9S0>
- Object Relational Mapper für node.js (ORM)
<https://www.npmjs.com/package/orm>
- Sequelize ORM für node.js <http://docs.sequelizejs.com>

(Diese Quellen wurden im Unterricht ausgeteilt bis auf Sequelize)

OWASP

- “The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software. Our mission is to make software security visible, so that individuals and organizations worldwide can make informed decisions about true software security risks.
- Everyone is free to participate in OWASP and all of our materials are available under a free and open software license.”

Quelle: https://www.owasp.org/index.php/Main_Page

Top10 (im Unterricht besprochen A1, A3, A5) :

- A1 Injection
- A2 Broken Auth
- A3 XSS
- A4 Insecure DOR
- A5 Misconfig
- A6 Sensitive Data
- A7 Access Controls
- A8 CSRF
- A9 Insecure Components
- A10 Redirects

Quellen/Texte zum Absichern von node.js Servern (neben OWASP)

- **OWASP Node Tutorial: Die 10 größten Fehler bei node.js**
<http://nodegoat.herokuapp.com/tutorial/a1>
- **Demo-App zum Testen der 10 Security-Fehler**
<https://github.com/OWASP/NodeGoat>

(Diese Quellen wurden im Unterricht nicht ausgeteilt)

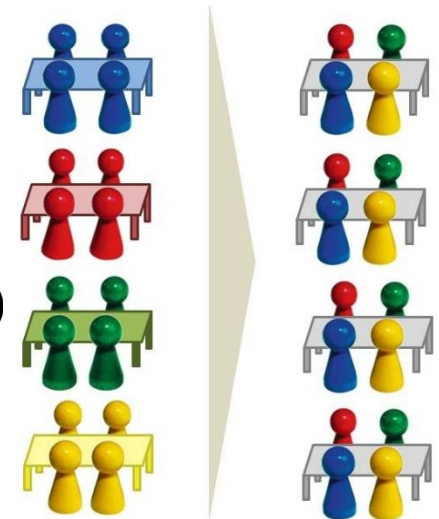
Quellen/Texte zum Skalieren von node.js (und 100% Uptime)

- **‘Uncaught Exception’ Handler: keine gute Idee**
https://nodejs.org/docs/latest/api/process.html#process_event_uncaughtexception
- **Besser einen Cluster an Node.js Instanzen nutzen**
<https://nodejs.org/api/domain.html>
- **Nginx und cluster kombiniert:** <http://cjhrig.com/blog/scaling-node-js-applications/>
- **Mit forever Server neustarten:**
<https://github.com/foreverjs/forever>

(Diese Quellen wurden im Unterricht ausgeteilt bis auf „forever“)

Expertengruppen: Ablauf

- **Phase 1: Bildung von Arbeitsgruppen zu den Themen (2min)**
 - A: mongoDB und Vergleich zu postgresQL
 - B: node.js app.js Aufbau und Absicherung
 - C: Closures und die Verwendung in der Praxis
- **Phase 2: Arbeitsgruppen werden zu Expertengruppen durch Textarbeit (10min), Recherche (5min) und Diskussion (10min)**
 - Was ist das?
 - Wie wendet man es an?
 - Wann?
 - Welche Grenzen/Nachteile gibt es?
 - ..
- **Phase3: Bildung neuer (Transfer)Gruppen, in welchen jeweils mind. ein Experte pro Thema sitzt (3x7=21min)**
 1. Nacheinander erläutern der wichtigsten Ergebnisse zu den W-Fragen
 2. Fragen aus der Runde klären
 3. Offene Fragen fürs Plenum notieren
- **Phase 4: Fragen im Plenum stellen und klären (10min)**



Agenda

- **Wiederholung zu Modularisierung**
 - **Wiederholung**
 - Closures
 - Warten auf Asynchrone Aufrufe (Callbacks)
 - Postman
 - **Vertiefung**
 - A: *SQL in node.js
 - B: node.js Absicherung (10 häufige Sicherheitslücken)
 - C: Skalierung von node.js Anwendungen
- **Ausblick**

Nächster Unterricht

- Datenhaltung
- NoSQL
- MongoDB
- mongoose für node.js

**Vielen Dank
und bis
zum nächsten
Mal**

HOW TO WRITE A CV



Leverage the NoSQL boom