

Übungsblatt 4

Veröffentlicht am	24.05.2016
Anzahl der Seiten	5
Anzahl Punkte im Pflichtteil (entspricht maximal erreichbaren Punkten)	15
Anzahl Punkte im Bonusteil	5
Abgabetermin und Demonstration in der Übung	Do, 09.06.2016

Anmerkungen

Bitte beachten Sie die folgenden Hinweise zur Bearbeitung der Übungsaufgaben und dem Ablauf im aktuellen Semester.

- Lesen Sie bei einem Übungsblatt stets alle Aufgaben durch, bevor Sie beginnen.
- Nach dem Unterricht wird vor der Übung das ggf. neue Übungsblatt in Moodle veröffentlicht.
<https://lms.beuth-hochschule.de/moodle/course/view.php?id=8190>
- Sofern zum Aufgabenblatt Code-Bausteine (Vorgaben) dazugehören, werden diese ebenfalls auf Moodle zum Download angeboten und sind als Ausgangsbasis bei der Bearbeitung zu verwenden.
- Ihre Lösung der Aufgaben laden Sie *vor* Ihrer persönlichen Demonstration in Moodle hoch. Dateiname: Ü[Nr]__Nachnamen__Matrikelnummern.ZIP
Beispiel: Ü2__Mueller_Meier__12345678__87654321.ZIP
- Erfordern die Teil-Aufgaben eines Übungsblattes, dass Sie mehrere Anwendungen, HTML-Seiten oder Code-Pakete erstellen, dann legen Sie bitte Unterordner in Ihrem ZIP mit den Nummern der Aufgaben an.
- Persönliche Demonstration und Erklärung in der Übung durch alle Gruppenmitglieder. Jedes Gruppenmitglied kann die Abgabe erläutern, sonst keine Punkte.
- Bei den Aufgaben ist jeweils angegeben, ob diese Pflicht- oder Bonus-Aufgaben sind, sowie die maximal erreichbaren Punkte der Teil-Aufgabe.
- Eine Übung gilt als bestanden, wenn mind. 50% der Pflichtpunkte erreicht wurden, sonst gibt es 0 (Null) Punkte.
- Bei verspäteter Abgabe von bis zu maximal 2 Wochen können nur noch 50% der möglichen Punkte des Übungsblattes erreicht werden (bei einer Woche verspäteter Abgabe 75% der möglichen Punkte).

Ziel und Zweck der Übung:

Nachdem Sie eine vorgegebene REST-Schnittstelle erweitert haben (Übung 3), können Sie nun eine eigene Implementierung basierend auf einer vorgegeben Definition implementieren. Sie implementieren ihren REST Server basierend auf gegebenen Testdefinitionen (in mocha.js). Modulare Entwicklung soll eingesetzt werden, um Ihren Code übersichtlich zu halten.

Die hier zu implementierende REST-Schnittstelle dient als Basis für die später folgende Anbindung einer Datenbank und das Abrufen von Client-Seite zur Anzeige einer dynamischen Webseite für Videos.

In dieser Übung sind keine HATEOAS-Erweiterungen wie href: oder items: Felder zu implementieren!

Referenzen:

- HTTP-Statuscodes bei Wikipedia (<https://de.wikipedia.org/wiki/HTTP-Statuscode>)
- Foliensatz SU3 und SU4 zu REST in node.js
- Foliensatz SU5 zum Testen (mocha.js), Foliensatz SU6 zum Modularisieren (CommonJS in node.js).
- Zum mocha.js Tests lesen und verstehen:
 - <https://mochajs.org/#exclusive-tests> um kurzzeitig nur einzelne Tests durchzuführen
 - <https://github.com/tj/should.js>
 - <https://www.npmjs.com/package/should-http>
 - <https://github.com/visionmedia/supertest>
- Um geschickt Variablen und Werte zwischen verschiedenen Middlewares und Handlern hin/herzureichen, verwenden Sie bspw. `response.locals` Variablen, siehe <http://expressjs.com/api.html#res.locals>

Vorbereitung (keine Punkte)

1.) Laden Sie das Codepack für das Übungsblatt herunter und führen Sie nach dem entpacken `npm install` aus. Installieren Sie mit `npm install -g mocha` das Testframework mocha. Sie können nun mitgelieferte Tests mittels `npm test` ausführen. Die Tests im Verzeichnis `./tests/` sind nach Aufgaben gruppiert. Verschieben Sie einzelne Dateien kurzzeitig nach `./tests-disabled/`, um nur bestimmte Tests laufen zu lassen.

2.) Setzen Sie für Sichtbarkeit von Fehlern die Umgebungsvariablen

- `NODE_ENV = development`
- `DEBUG = me2u4:*`

3.) Passen Sie in der Datei `./test/config_for_tests.js` Ihren BasisURL Pfad an, wenn der nicht <http://localhost:3000> ist.

4.) Für WebStorm ist im Codepack bereits eine Startkonfiguration für mocha mit dabei. Passen Sie ggf. die Pfade für sich an (Run->Edit Configurations -> mocha -> Mochatest U4. Wer nicht mit WebStorm arbeitet kann auch in anderen IDEs mocha dort aufrufen...oder eben über die Konsole mit „mocha“ oder „npm test“.

Tipp: Es empfiehlt sich nicht ständig alle Tests laufen zu lassen. Kopieren Sie z.B. alle Tests, die sie gerade nicht brauchen in ein anderes Verzeichnis. Dann läuft der verbleibende Test schneller und die Fehlermeldung ist leichter zu finden. Schauen Sie unbedingt auch auf die *Konsolen-Ausgaben Ihrer Server-App*, nicht nur auf die Fehlermeldung bei mocha. Sonst verpassen Sie vielleicht wertvolle Stacktrace-Ausgaben Ihres Servers.

Die `empty_test.js` sollte direkt nach dem Entpacken und `npm install` laufen. Alle weiteren Tests schlagen am Anfang noch fehl. Viel Erfolg!

Aufgabe 1 (Pflicht, 6 Punkte insgesamt)

In dieser Aufgabe entwickeln Sie die CRUD-REST-Operationen für eine neue Ressourcensammlung namens `videos`. Videos sollen folgende Attribute haben

- `id` (Number, von Außen nicht setzbar, automatisch bei POST)
- `title` (String, required)
- `description` (String, optional, default „“)
- `src` (String, required)
- `length` (Number; nicht negative Zahl für Sekundenangabe, required)
- `timestamp` (Number, nicht von Außen setzbar, automatisch bei POST)
- `playcount` (Number; nicht negative Zahl, optional, default 0)
- `ranking` (Number; nicht negative Zahl, optional, default 0)

Aufgabe 1.a (Pflicht, 3 Punkte)

Implementieren Sie das Modul `videos`, welches bisher nur als unfertige Datei in `./routes/videos.js` liegt. Das Modul soll alle CRUD-Routen in einer `express.Router()` Instanz registrieren und diesen Router als Modul-Export zurückliefern. BEI POST und PUT soll das gesamte gespeicherte Objekt zurückgeliefert werden.

Nutzen Sie zum Abrufen und Speichern Ihrer JSON-Objekte wieder die `store.js` (wie in Übung 3, nur eine neue Version davon). Achten Sie auf die korrekten Status-Codes bei der Rückgabe:

- **200** Suche erfolgreich, normale Rückgabe
- **201** Element erzeugt
- **204** (No Content) Einfache Rückgabe nach dem Löschen eines Elementes
- **400** Unzureichende Anfrage (z.B. Parameter fehlen oder sind falsch)
- **404** Route nicht gefunden, Element nicht gefunden
- **405** die HTTP-Methode ist nicht erlaubt auf dieser URL
- **406** Anfrage kann nicht erfüllt werden (z.B. falscher Datentyp für Antwort verlangt)
- **415** Falscher Datentyp an Server gesendet

Binden Sie Ihren Router in Server-App mittels `app.use(...)` ein und testen Sie die Funktionalität mit dem bereitgestellten tests (`npm test`¹).

Aufgabe 1.b (Pflicht, 3 Punkte)

Fehlende optionale Felder sollen den default-Wert bekommen.

Alle required-Felder sollten vor dem Speichern überprüft werden und ein HTTP-Statuscode **400** zurückgegeben werden, wenn davon eines fehlt.

Generell soll bei allen Fehlern im `body` ein JSON-Fehlerobjekt wie `{ „error“: { „message“: „xyz“, „code“: 400 } }` gesendet werden (mit gesetztem Code). (Diese JSON-Error-Objekte gab es bereits in Übungsblatt 3, siehe Musterlösung 3)

¹ Mocha-Tests können Sie auch integriert direkt in WebStorm starten. Nutzen Sie dazu eine WebStorm Mocha-Run-Konfiguration für das Test-Verzeichnis `./test`

Aufgabe 2 (Pflicht, 9 Punkte insgesamt)

Mittels Filterangaben kann die Rückgabe durch den Client seinen Bedürfnissen nach beeinflusst werden. Dafür dienen u.a. GET-Parameter wie **filter**, **offset**, **limit** und auch Suchparameter.

Tipp: Um auf JavaScript-Objekten über alle eigenen Objekteigenschaften zu iterieren, um diese ggf. zu entfernen, können Sie bspw. folgenden Code verwenden:

- https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Object/keys

Auf einem Array (wie bspw. den Eigenschaften) können Sie mittels Filterfunktionen oder `forEach` operieren.

- https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
- https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

Um leicht von einem JSON-Objekt eine Referenzfreie, tiefe Kopie zu erhalten, können Sie

```
var copy = JSON.parse(JSON.stringify(myVariable));
```

benutzen.

Aufgabe 2.a (Pflicht, 4.5 Punkte)

Unterstützen Sie die Rückgabe nur bestimmter Attribute Ihrer **videos**. Wenn ein GET den Parameter **filter** enthält, liefern Sie nur die Attribute, die in **filter** angegeben sind. Beispiel: `?filter=title,src` liefert das eine (oder mehrere) Video-Objekte nur mit diesen Attributen. Gibt es ein Attribut gar nicht in der Ressourcensammlung, geben Sie einen Fehler **400** zurück (siehe 1.b).

Empfehlung: Implementieren Sie Ihre Filter-Lösung als Middleware und binden Sie ein. Orientieren Sie sich bspw. an `./rest-api/request-checks.js`

Aufgabe 2.b (Pflicht, 4.5 Punkte)

Implementieren Sie das Blättern im Ergebnis mit den GET-Parametern **limit** und **offset**. Beispiel: `?limit=5&offset=2` liefert die Videos `[2,3,4,5,6]`, also die Videos 3-7. Denken Sie an eine Überprüfung ungültiger Werte.

Empfehlung: Auch hier kann die Lösung über eine Middleware erfolgen.

Aufgabe 3 (Bonus, 2.5 Punkte insgesamt) - Suchen und Patchen

Bonusaufgaben sind nicht durch die mocha-Tests abgedeckt!

Aufgabe 3.a (Bonus, 1.5 Punkt)

Implementieren Sie das Suchen nach bestimmten Feldwerten. Alle GET-Parameter, die es als Attribute in Ihrer Ressourcensammlung gibt, geben zu enthaltene Werte an. Beispiel: `?title=Beuth&description=2015` liefert ein (oder mehrere) Videos, die im `title` den String `Beuth` enthalten und in der `description` den String `2015`. Beachten Sie die möglichen Wechselwirkungen mit Ihrem Code zu 2.a und 2.b. Gibt es ein Attribut gar nicht in der Ressourcensammlung, geben Sie einen Fehler `400` zurück (siehe 1.b).

Aufgabe 3.b (Bonus, 1 Punkt)

Implementieren Sie eine PATCH-Methode für `/videos/:id`, die nicht idempotent ist und welche das Inkrementieren von `playcount` erlaubt. Format eingehender JSON-Objekte `{„playcount“: „+1“}`.

Aufgabe 4 (Bonus, 2.5 Punkte insgesamt)

Aufgabe 4.a (Bonus, 1 Punkt)

Legen Sie eine zweite Ressourcensammlung `comments` in einer `n:1` Beziehung zu `videos` an mit folgenden Attributen:

- `id` (Number, nicht von Außen setzbar, automatisch bei POST)
- `videoid` (Number, required)
- `text` (String, required)
- `timestamp` (Number, nicht von Außen setzbar, automatisch bei POST)
- `likes` (Number; nicht negative Zahl, optional, default 0)
- `dislikes` (Number; nicht negative Zahl, optional, default 0)

Achten Sie auch hier auf die Prüfung der required-Felder.

Aufgabe 4.b (Bonus, 0.5 Punkt)

Prüfen Sie beim Anlegen eines `comments`, dass die `videoid` existiert. Löschen Sie beim Entfernen eines Videos alle damit verbundenen `comments` ebenfalls.

Aufgabe 4.c (Bonus, 1 Punkt)

Unterstützen Sie alle Ihre implementieren Filter aus Aufgabe 2 (a-b) auch für `comments`. Finden Sie eine Implementierungslösung, bei der der Code nicht komplett dupliziert werden muss (kein copy&paste), sondern an einer „zentralen Stelle“ die Filterunterstützung für ihre beiden Ressourcensammlungen programmiert ist.