BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

# Übungsblatt 5

| | |
|---|---|
| Veröffentlicht am | 07.12.2016 |
| Anzahl der Seiten | 4 |
| Anzahl Punkte im Pflichtteil<br>(entspricht maximal erreichbaren Punkten) | **10** |
| Anzahl Punkte im Bonusteil | 5 |
| Abgabetermin und Demonstration in der Übung | Übungen 10./12.01.2017 |

**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.
- 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 verspätete 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).

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

## Ziel und Zweck der Übung:

Sie nutzen die MongoDB zum Speichern und Ausliefern Ihrer Ressourcensammlungen einer REST-Schnittstelle. Sie erkennen die Vorteile der JSON-basierten Speicherung und des Abrufes im Vergleich zum bekannten mySQL. MongoDB Dokumente entsprechen einzelnen Ressourcen. Zur Absicherung der Datenkonsistenz nutzen Sie `mongoose` und können die Konzepte des Schemas, der Validierung/Prüfung und der Models nutzen.

**Referenzen:**

- Foliensatz SU8 zu NoSQL, mongoDB und `mongoose`
- mongoDB Doku https://docs.mongodb.org/manual/
- mongoose Doku http://mongoosejs.com/docs/guide.html
- mongoose `schema.paths` http://stackoverflow.com/questions/17035297/getting-schema-attributes-from-mongoose-model

## Vorbereitung (keine Punkte)

1. Installieren Sie mongoDB entsprechen der Anleitung aus den Unterrichtsfolien.
2. Nutzen Sie entweder Ihre Lösung zu Übungsblatt 4 (Kopie machen) oder laden Sie das Codepack für das Übungsblatt 5 herunter und führen Sie nach dem Entpacken `npm install` aus.
3. Installieren Sie danach mit `npm install --save mongoose` das Modul zur Schema/Model-basierten Kommunikation mit ihrer mongoDB.
4. Starten Sie in einem Terminal-Fenster aus Ihrem Verzeichnis des Projektes (bspw. `C:\Uebung\Uebung5`) die mongoDB mit `mongod --dbpath ./mongodb-data`[1] und vergewissern Sie sich, dass die mongoDB gestartet ist (auf Verbindungen wartet).
   Schließen Sie das Terminalfenster nicht! (sonst ist mongoDB beendet).

**MongoDB läuft jetzt und Sie können mit dem Erstellen Ihrer Lösung beginnen.** Die URL zum Verbinden mit der mongoDB lautet üblicherweise `mongodb://localhost:27017/me2`

*Hinweis:* Sie können statt `me2` auch einen anderen Datenbank-Namen wählen.
*Hinweis:* Möchten Sie alle Daten löschen, dann beenden Sie `mongod` (`Strg+C`), löschen alle Dateien in `./mongodb-data` und starten wieder neu.

[1] Sollten Sie eine Fehlermeldung erhalten, dass mongod.exe nicht gefunden werden kann, dann müssen Sie das Installationsverzeichnis von mongoDB ihrem Suchpfad in den Umgebungsvariablen hinzufügen. *Systemsteuerung->Umgebungsvariablen bearbeiten*. Bei `PATH` mit Semikolon (;) abgetrennt den Pfad zum bin-Ordner (bspw. `C:\Program Files\MongoDB\Server\3.0\bin` einfügen). Terminalfenster danach neu öffnen. Das Verzeichnis `./mongodb-data` muss vor dem Starten der DB angelegt worden sein.

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

## Aufgabe 1 (Pflicht, 5 Punkte insgesamt)

In dieser Aufgabe erstellen Sie zunächst ein *Mongoose Model* für ihre Videos und binden die DB-Dokumentensammlung an Ihre REST-Schnittstelle an. Hier nochmal die Angaben dazu (identisch mit Übungsblatt 4 bis auf _id und timestamp).

- `_id` (String, von Außen nicht setzbar, automatisch bei POST)
- `title` (String, required)
- `description` (String, optional, default '' [leerer String])
- `src` (String, required)
- `length` (Number; nicht negative Zahl für Sekundenangabe, required)
- `timestamp` (String, 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, 2 Punkte)

Legen Sie im Projekt einen Ordner `models` an und darin eine neue Datei `video.js` . Definieren Sie darin das entsprechende *Mongoose Schema* und als Modul-Export das passende *Mongoose Model* für **`Video`**.

- Notwendige Felder und optionale Felder, sowie Default-Werte und Mindest-Wert bei Zahlen sollen berücksichtigt werden.
- *Hinweis:* Damit MongoDB das Attribut mit dem Erstellungszeitpunkt nicht `createdAt` sondern `timestamp` nennt, nehmen Sie die Option für `timestamps` aus den Unterrichtsfolien dazu (siehe dort Mongoose Schema)

### Aufgabe 1.b (Pflicht, 2 Punkte)

Binden Sie die Moduldatei `./models/video.js` in Ihre node.js-Anwendung ein.
Implementieren Sie nun den REST-Handler für HTTP POST auf `/videos` .
*Hinweis:* Legen Sie zunächst eine neue Model-Instanz an und übergeben der Konstruktorfunktion direkt `req.body`. Sobald Sie auf Ihrer Model-Instanz ein `.save(..)` durchführen, wird das Schema geprüft und Fehler werden ggf. zurückgegeben. Versäumen Sie daher nicht `.save(..)` eine Callbackfunktion mitzugeben, in welcher Sie die Fehler erkennen und zurückmelden oder bei Erfolg das gespeicherte Objekt zurückliefern. Statuscode `201` nicht vergessen.
Ein Fehler wird (wie in Übung 4) wieder als JSON-Fehlerobjekt `{ „error“: { „message“: „xyz“, „code“: 400}` zurückgeliefert und parallel als passender HTTP-Statuscode gesetzt.
**Zum Abschluss von Aufgabe 1.b** führen Sie mindestens zwei erfolgreiche POST Requests durch.

### Aufgabe 1.c (Pflicht, 1 Punkt)

Fügen Sie den erforderlichen Programmiercode für ein GET der Ressourcensammlung `/videos` hinzu, welcher alle Video-Dokumente aus der MongoDB Collection `videos` zurückliefert.

## Aufgabe 2 (Pflicht, 5 Punkte)

Implementieren Sie nun auch die REST-Anbindung für `/videos/:id`
(Methoden GET, PUT, PATCH, DELETE).

- Prüfen Sie dabei (wie in Übung 4) für PUT und PATCH, dass die `_id` aus dem zugesendeten JSON-Datensatz mit der ID aus der Ressourcen-URL übereinstimmt, bevor Sie speichern (Bei PATCH nur, wenn `_id` mit gesendet wurde vom Client, fehlt es, dann ist das bei PATCH hier auch ok).
- Achten Sie auch darauf, dass Sie kein Überschreiben von `_id` und `__v` erlauben. Ignorieren Sie diese Angaben einfach beim Speichern in die DB.

*Hinweis für PUT und PATCH:* Mit der Mongoose Model-Methode `.findByIdAndUpdate(..)` ist ein PATCH sehr leicht zu implementieren. Für ein PUT können Sie probieren, ob Sie eine neue Instanz des Models zu erstellen (wie bei POST in Aufgabe 1.b) oder mittels der Eigenschaft `.schema.paths` des Mongoose Models durch die Attribute und Typen des Schemas durchgehen (um diese in `req.body` zu prüfen), bevor Sie die Datenbankinhalte aktualisieren. Die Dokumentation dazu entnehmen Sie den Referenzen oben. Insbesondere sollten fehlende optionale Felder bei einem PUT auch anschließend in der Datenbank wieder auf dem Default-Wert stehen.

## Aufgabe 3 (Bonus, 2 Punkte insgesamt)

Sie werden bemerkt haben, dass das Feld updatedAt durch mongoose nicht automatisch auf das aktuelle Datum bei POST und PATCH gesetzt wurde, wenn Sie `.findByIdAndUpdate(..)` verwenden. Das funktioniert nur bei einem `.save()`. Aktualisieren Sie daher das Update-Datum selbst beim Speichern. Prüfen Sie vorher auch, ob das zugesendete `updatedAt`-Datum der JSON-Daten bei einem PUT noch identisch ist mit dem aktuell gespeicherten, sonst lehnen Sie den PUT mit einem Statuscode `409` (Conflict) ab[2].

## Aufgabe 4 (Bonus, 1.5 Punkte insgesamt)

Implementieren Sie den HTTP GET-Parameter `filter` wie bereits im Übungsblatt 4, Aufgabe 2.a beschrieben. Diesmal überlassen Sie die Filterung jedoch direkt der Datenbank und weisen Mongoose an, nur die gewünschten Attribute zu liefern (siehe Unterrichtsfolien). `_id` darf dabei weiter in den zurückgelieferten Dokumenten mit vorkommen.

## Aufgabe 5 (Bonus, 1.5 Punkte insgesamt)

Implementieren Sie die GET-Parameter `offset` und `limit` (auch gleichzeitige Verwendung möglich) wie bereits im Übungsblatt 4, Aufgabe 2b beschrieben. Diesmal überlassen Sie das Heraussuchen der relevanten Dokumente direkt der Datenbank und weisen Mongoose entsprechend direkt in der Anfrage an, `offset`[3] und `filter` zu berücksichtigen.
*Prüfen Sie:* Was passiert, wenn ungültige Werte für `offset` und/oder `limit` angegeben werden? Müssen Sie dies weiterhin vorher prüfen oder kommt mongoose bspw. auch mit einem `offset` von `-1` und einem `limit` von "`noLimit`[4]" zurecht und liefert eine passende Fehlermeldung?

[2] Der anfragende Client hat dann offenbar eine veraltete Version der Daten und müsste erst per GET aktualisieren
[3] `offset` wird bei mongoDB `skip` genannt
[4] das wären als URL-GET-Parameter `?offset=-1&limit=noLimit`