

Übungsblatt 3

Veröffentlicht am	03.05.2016
Anzahl der Seiten	3
Anzahl Punkte im Pflichtteil (entspricht maximal erreichbaren Punkten)	10
Anzahl Punkte im Bonusteil	5
Abgabetermin und Demonstration in der Übung	Do, 19.05.2016

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:

Basierend auf dem Quellcode zur Erstellung von REST-Schnittstellen in node.js, welcher im Unterricht besprochen wurde, ergänzen Sie weitere Routen und testen Ihre Erweiterungen (vorzugsweise mit Postman). Nach Abschluss der Übung sollen Sie die Verbindung von HTTP-Methoden (Operationen) und URLs für Routen (Ressourcen) mit node.js nachvollziehen und umsetzen können.

Erweiterte Lernziele sind der Umgang mit fehlerhaften Anfragen, sowie Überprüfung der Parameter und HTTP Header Felder zur Client-Server-Kommunikation.

Referenzen:

- Foliensatz zu Unterricht „REST APIs“, sowie „REST in node.js“
- **Codepaket zum Übungsblatt 3 aus Moodle**
- RegExp in JavaScript
https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/RegExp
- URL-Abschnitte eines Request wiederverwenden: <http://stackoverflow.com/questions/10183291/how-to-get-the-full-url-in-express-js>
bzw. auch: <http://expressjs.com/api.html#req>
- Methoden `forEach(..)` und `filter(..)` bei JavaScript Arrays: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array

Vorbereitung (keine Punkte)

Laden Sie das Codepaket zum Übungsblatt 3 aus dem Moodlekurs herunter. Es dient als Basis für die folgenden Aufgaben.

Das Codepaket enthält u.a. das minimalistisch gehaltene Modul `store`, welches in der `app.js` bereits als Variable `store` angelegt wird. `store` bietet Ihnen vier Methoden an, damit sie CRUD-Operationen durchführen können.

- **`select`** (String type, Number id)
@returns undefined, genau ein Element, wenn id gegeben wurde, oder ein Array von Elementen
- **`insert`** (String type, Object element)
@returns id des neu eingefügten Elementes]
- **`replace`** (String type, Number id, Object element)
@returns this (das store Objekt)
- **`remove`** (String type, Number id)
@returns this (das store Objekt)

Beachten Sie: `store` enthält bereits JSON-Objekte für zwei *types* (Ressourcensammlungen): `tweets` und `users`. Es sind `tweets` mit der id 101 und 102, sowie `users` mit der id 103 und 104 bereits vorhanden. Bei jedem Neustart Ihrer Anwendung werden alle Änderungen an `store` vergessen und es existieren wieder nur diese vier Objekte (*Es handelt sich also um eine in-memory-Datenbank*). Sie können als type jeden beliebigen String verwenden (bspw. „retweets“, „friendships“, „tags“, ...). JSON-konforme Objekte für den `store` müssen ein Attribut `id` besitzen. Schauen Sie sich die Beispiele in der Datei `.\blackbox\store.js` an (Zeilen ca. 25-50). Ergänzen Sie den Code, wenn Sie weitere Datensätze beim Neustart bereits in `store` haben möchten.

Aufgabe 1 (Pflicht, 4 Punkte insgesamt)

Derzeit existiert nur eine REST-API für `tweets`. Erweitern Sie die aktuelle API um eine weitere Ressourcensammlung (Ressource Collection). Ihre neue Ressourcensammlung soll eine 1:n oder n:1 Beziehung zu `tweets` haben.

Aufgabe 1.a (Pflicht, 1 Punkt)

Wählen Sie eine neue, passende URL-Repräsentation für die neue Ressourcensammlung. Legen Sie dafür mindestens eine Route (mit Handler-Funktion) in Ihrer `node.js` Anwendung an.

Aufgabe 1.b (Pflicht, 3 Punkt)

Legen Sie weitere Route-Handler für Ihre neue Ressourcensammlung an, damit alle relevanten CRUD-Operationen REST-konform (Level 2) unterstützt werden. Führen Sie die Funktionalität mittels Postman bei der Abgabe vor.

Aufgabe 2 (Pflicht, 6 Punkte insgesamt)

Zur Erinnerung: Ihre neue Ressourcensammlung aus Aufgabe 1 soll eine 1:n oder n:1 Beziehung zu tweets haben.

Aufgabe 2.a (Pflicht, 3 Punkte)

Ergänzen Sie Ihren app.js-Code so, dass bei einem HTTP GET die zurückgelieferten JSON-Repräsentationen für jedes Objekt auch das Attribut href mit der passenden URL besitzen, welche dieses Objekt in der REST-Schnittstelle repräsentiert. **Achtung:** In den Fällen, in denen Sie ein Array an Elementen zurückliefern, müssen Sie Ihre JSON-Repräsentation so ändern, dass das Array bspw. als Wert des Attributes items ausgegeben wird, so dass Sie auch ein href-Attribut für dieses Array parallel zu items angeben können¹.

Das href-Attribut soll bei jeder Anfrage dynamisch durch Ihren Route-Handler zum Rückgabe-Inhalt hinzugefügt werden und nicht fest in den store.js reinprogrammiert werden!

Aufgabe 2.b (Pflicht, 3 Punkte)

Wenn Sie sich für eine n:1 Beziehung ihrer neuen Ressourcensammlung zu tweets entschieden hatten, liefern Sie bei der JSON-Antwort auf HTTP GET http://localhost:3000/tweets/:id für jeden tweet dynamisch einen Verweis auf die neue Ressourcensammlung mit. Wenn Sie sich stattdessen für eine 1:n Beziehung ihrer Ressourcensammlung zu tweets entschieden hatten, entsprechend anders herum¹.

Auch diese Verweise werden zu den Rückgabedaten dynamisch ergänzt und nicht fest in den store.js reinprogrammiert.

Aufgabe 3 Bonusaufgabe (Optional, 2.5 Punkte)

Ergänzen Sie den Code der REST-Schnittstelle um eine Unterstützung für HTTP PATCH bei einer der beiden Ressourcensammlungen. Ist Ihre Lösung idempotent? Begründen Sie bei Abgabedemonstration Ihre Entscheidung für oder gegen eine idempotente Lösung.

Aufgabe 4 Bonusaufgabe (Optional, 2.5 Punkte)

Sofern Sie Aufgabe 2.b gelöst haben, ergänze Sie den Code der REST-Schnittstelle um eine Unterstützung für den GET-Parameter expand (Bei einer 1:n Beziehung wäre das ein ?expand=tweets und bei n:1 Beziehung entsprechend mit der Angabe Ihrer Ressourcensammlung). Damit liefert ein HTTP GET neben der angefragten Ressource auch gleich die dazugehörige Ressourcensammlung aus der 1:n Beziehung mit¹.

¹ Ein Beispiel finden Sie in den Unterrichtsfolien zu HATEOAS mit expand-Unterstützung.