Übungsblatt 11: Node.js

Vorstellung in den Tutorien am 15. Januar 2015

Hinweis: In dieser Übung geht es hauptsächlich darum JavaScript in der Node.js Laufzeitumgebung auszuführen. Als Voraussetzung sollten Sie Aufgabe 5 von Übung 10 durchgeführt haben, mindestens jedoch Node.js lokal bei sich installiert haben.

Zum Erstellen der JavaScript-Dateien verzichten wir auf die Vorgabe eines bestimmten Cloud-Tools. Sie dürfen mit einem Editor Ihrer Wahl arbeiten. Theoretisch kann jeder normale Texteditor für diese Aufgaben benutzt werden. Allerdings ist es sehr praktikabel spezialisierte Entwicklungsumgebungen für die Programmierung zu benutzen. Webstorm[1] wird zum Beispiel für Studenten kostenlos angeboten. Andere gute Editoren sind: Sublime Text, Atom, Brackets.

Die Vorführung der Aufgaben im Tutorium kann entweder mit eigenem Laptop erfolgen, oder indem die Dateien auf dem Laptop des Tutors ausgeführt werden.

Zum Nachschlagen von JavaScript ist neben den Vorlesungsfolien das Mozilla Developer Network[2] hilfreich. Wenn Sie öfter in JavaScript programmieren bzw. kooperativ an einem Projekt arbeiten, empfehlt es sich eine Konvention einzuhalten. Unter folgender URL sind einige Empfehlungen aufgelistet: `https://github.com/rwaldron/idiomatic.js/`

11.1 Datei-Server mit Node.js (3 Punkte)

Wir beginnen nun mit der Erstellung eines eigenen Webservers. Im Gegensatz zu den bekannten Webservern Apache und nginx ist Node.js eine Runtime für JavaScript. Das bedeutet, dass wir erst einen Webserver programmieren müssen, bevor ein Client Dateien (HTML, CSS, JS) per HTTP abfragen kann.

1. Als Folgeaufgabe zu Übung 10.5, lesen Sie nun die restlichen Abschnitte von Kapitel 20 des Buches Eloquent JavaScript[3] durch (ab Abschnitt „The File System Module“). Vollziehen Sie die Schritte zur Erstellung des Datei-Servers nach und setzen Sie ihn um. Testen Sie Ihre Implementierung mit curl (oder dem Tool aus Hinweis 1). Achten Sie darauf den richtigen Port in der URL anzugeben.

2. Der vorgegebene Webserver kann zwar Verzeichnisse löschen, aber keine Neuen erstellen. Fügen Sie im Code eine Methode `MKDIR` hinzu, mit der Sie ein neues Verzeichnis anlegen. Achten Sie darauf, dass diese Methode idempotent ist.

Hinweis 1: Wenn Sie das genannte Commandline-Tool *curl* nicht nutzen wollen, eignet sich zum Testen auch jeder grafische REST-Client. Empfehlenswert ist u.a. die Chrome App Postman[4].

[1] `https://www.jetbrains.com/webstorm/`
[2] `https://developer.mozilla.org/de/docs/Web/JavaScript`
[3] `http://eloquentjavascript.net/20_node.html`
[4] `https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop`

11.2 Webseite zur Benutzung des Datei-Servers (4 Punkte)

Mit dem zuvor erstellten Datei-Server[5] ist es möglich beliebige Dateien mit richtigem *Content-Type*-Header auszuliefern. Erstellen Sie in dieser Aufgabe eine eigene Webseite mit CSS- und JavaScript-Datei, welche der Server auf Anfrage zurück gibt. Der Zweck dieser Webseite soll es sein ihren Datei-Server mittels einer grafischen Oberfläche zu bedienen.

Ihre Webanwendung muss die folgenden Anforderungen erfüllen:

- Die Webseite soll alle Funktionalitäten des Servers (Datei/Verzeichnis lesen, erstellen und löschen) abbilden. Dafür müssen für die jeweiligen Methoden entsprechende Buttons und Eingabefelder zur Verfügung stehen, z.B. um den Dateinamen und den Inhalt festzulegen/anzusehen.
- Die Interaktion zwischen der an den Browser ausgelieferten Webseite und dem Server soll in JavaScript mittels `XMLHttpRequest`-Objekt im Hintergrund erfolgen.
- Die URL ihrer Webanwendung ändert sich nie.
- Die gesamte Übertragung der Daten zwischen Server und Client soll in JSON formatiert sein. (Erfordert eine Änderung des Server-Codes.)
- Wenn ein Verzeichnis abgefragt wird, sollen alle Dateien gelistet werden und per Maus auswählbar sein. Dadurch wird eine bestimmte Datei selektiert und der Inhalt angezeigt.
- Strukturieren Sie Ihre Seite mittels CSS übersichtlich und benutzerfreundlich.

Hinweis: Achten Sie darauf, dass es nicht möglich sein soll die Dateien für Ihre „Basis“-Webseite zu verändern. Bei einem Reload könnte es ansonsten passieren, dass keine Webseite mehr ausgeliefert wird, weil Sie diese gelöscht haben.

[5] Falls Sie die vorherige Aufgabe nicht gemacht haben, verwenden Sie den Sourcecode von dieser Seite: `http://eloquentjavascript.net/code/file_server_promises.js`

11.3 Content Negotiation (3 Punkte)

REST besagt, dass jede Ressource durch unterschiedliche Repräsentationen dargestellt werden kann. In HTTP ist dieses Prinzip mittels *Content Negotiation*[6] umgesetzt.

1. Entwickeln Sie einen Webserver in Node.js, der Ihnen ihre Personendaten (Name, Jobbezeichnung, E-Mail) bei einem HTTP-Request mit spezifizierten *Accept*-Header in dem angegebenem Format zurück gibt. Mindestens unterstützt werden muss `text/plain`, `text/html` und `application/json`. (Als Beispiel können Sie die Ressource `http://eloquentjavascript.net/author` mit Postman nach unterschiedlichen Repräsentationen abfragen. Ihr Webserver soll die selbe Funktionalität bieten.)

2. Schreiben Sie durch Verwendung von Node's `http.request` Funktion einen Client, der Ihren zuvor geschriebenen Server anspricht und Ihre Personendaten in jedem möglichen Format abfragt. Die Antworten sollen auf der Console ausgegeben werden.

11.4 Wiederholung: Objektorientierung (3 Punkte)

In dieser Aufgabe werden Sie die Konzepte der Objektorientierung in JavaScript wiederholen. Lesen Sie sich dazu Kapitel 6 des Buches Eloquent JavaScript[7] durch.

Bereiten Sie eine Präsentation oder ein ausführliches Quellcode-Beispiel vor, in welcher Sie die folgenden Konzepte/Begriffe erklären: *Kapselung, Prototypen, Konstruktoren, Prototyp-Interferenz, Prototypen-lose Objekte, Polymorphismus, Getters und Setters, Vererbung bei Typen/Konstruktoren*. Gehen Sie bei ihrer Präsentation mindestens auf die folgenden JavaScript Funktionalitäten ein: `Object.getPrototypeOf`, `in`–Operator, `Object.defineProperty` (mit jeder möglichen Definition/Option), `hasOwnProperty`, `Object.keys`, `instanceof`.

Zeigen Sie von Ihren Beispielen die vollständigen Prototypen-Ketten.

[6] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation`

[7] `http://eloquentjavascript.net/06_object.html`