A3 Web-Labor - Bookstore NodeJS Webserver (GK+EK)

Done

Voraussetzungen

- installierte Node.js-Umgebung inkl. npm-Paketmanager: https://nodejs.org/en/download/

Aufgabe

Node.js ermöglicht es, JavaScript nicht nur im Browser des Clients, sondern auch am Server zu verwenden. Mit Node.js können sehr einfach Webanwendungen sowie Webservices erstellt werden, die dann ein beliebiger Client (anderer Webserver, mobile Anwendung, Desktop-Anwendung, ...) ansprechen kann.

Entwickle im Rahmen eines kleinen Projekts einen einfache n Node.js-Webserver für eine Buchhandlung!

NodeJS Grundlagen

Nach der Installation von NodeJS können JavaScript-Dateien (.js) direkt über den Befehl "node" in der Konsole ausgeführt werden:

```
> node bookstore.js
```

Folgendes HalloWelt-Programm gibt den Text "Hallo Welt" in der Konsole aus, wenn die Datei über den Befehl "node" ausgeführt wird:

```
console.log("Hallo Welt!")
```

Ein großer Vorteil von NodeJS ist, dass ein einfacher Webserver über wenige Zeilen Code erstellt werden kann:

```
var http = require('http')

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
}).listen(8080);
```

Führt man das obige Programm in der Konsole aus und öffnet den Browser unter localhost:8080, sieht man einen ersten HalloWelt-Webserver. Die Methode http.createServer erzeugt einen neuen Webserver, der bei einem Aufruf durch einen Client "Hallo Welt" als Plaintext zurückliefert. Über listen wird der Server schließlich gestartet, sodass er auf eingehende Verbindungen "horcht".

Die Parameter "req" und "res" stehen dabei für Request bzw. Response. In req finden sich alle Daten, die vom Client geschickt wurden, während man über den res-Parameter die Antwort an den Client vorbereitet bzw. abschickt.

Über req.url erhält man Zugriff auf alle wichtigen Attribute der angegeben URL - es empfiehlt sich, diese vorher über url.parse zu parsen:

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    var u = url.parse(req.url, true)
    console.log(u.query)
    console.log(u.pathname)
    res.end('Hello World!');
  }).listen(8080);
```

Nach dem Parsen kann über u.query auf alle GET-Parameter der Anfrage zugegriffen werden, während u.pathname den gewählten Sub-Pfad abbildet. Ruft man beispielsweise die URL "http://localhost:8080/test?hallo=welt&x=y" auf, lautet die Ausgabe:

```
{ hallo: 'welt', x: 'y' }
/test
```

Die Parameter req und res enthalten noch weitere interessante Eigenschaften der HTTP-Anfrangen. Beispielsweise kann über req.method die verwendete HTTP-Methode (GET, POST, DELETE, PUT) abgefragt werden.

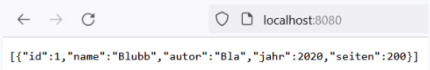
Nähere Informationen und Beispiele findet ihr unter https://www.w3schools.com/nodejs/

Anforderungen

Entwickle im Rahmen eines kleinen Projekts einen einfache n Node.js-Webserver für eine Buchhandlung!

GKÜ:

- Der Webserver liefert bei einer Client-Anfrage alle gespeicherten Bücher im JSON-Format zurück
- Ein Buch-Objekt besteht dabei aus folgenden Eigenschaften:
  - ID: Wird vom Server fortlaufend vergeben (beginnend mit 1)
  - Name: Name des Buchs
  - Autor: Autor des Buchs
  - Jahr: Jahreszahl der Veröffentlichung
  - Seitenanzahl: Anzahl der Seiten des Buchs



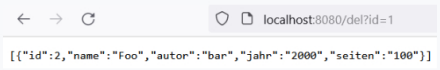

```
[{"id":1,"name":"Blubb","autor":"Bla","jahr":2020,"seiten":200}]
```

- Wird der Pfad "/add" aufgerufen, wird der Liste ein neues Buch hinzugefügt, wobei die Attributwerte für das neue Buch aus den GET-Parameter der URL ausgelesen werden. Beispiel: Die URL"localhost:8080/add?name=Foo&autor=bar&jahr=2000&seiten=100" fügt ein neues Buch mit dem Namen "Foo", dem Autor "Bar", das Jahr "2000" und die Seitenanzahl "100" hinzu. Als Rückgabe soll die neue (erweiterte) Liste wieder im JSON-Format zurückgeliefert werden.


```
[{"id":1,"name":"Blubb","autor":"Bla","jahr":2020,"seiten":200},{"id":2,"name":"Foo","autor":"bar","jahr":"2000","seiten":"100"}]
```

GKV:

- Bücher können auch gelöscht werden, indem der Pfad "/del" aufgerufen wird, wobei der GET-Parameter "id" verwendet wird, um das zu löschende Buch zu bestimmen. Als Ergebnis soll wieder die neue (reduzierte) Liste im JSON-Format zurückgeliefert werden. Beispielsweise löscht die URL "localhost:8080/del?id=1" das Buch mit der ID "1".




```
[{"id":2,"name":"Foo","autor":"bar","jahr":"2000","seiten":"100"}]
```

- Die Werte eines existierenden Buchs können geändert werden, indem der Sub-Pfad "/update" verwendet wird. Der GET-Parameter "id" gibt hierbei an, welches Buch geändert werden soll. Die restlichen GET-Parameter geben wie beim Hinzufügen die neuen Werte an. Die URL "localhost:8080/update?id=2&name=Blubb&autor=bla&jahr=2000&seiten=100" ändert das Buch mit der ID "2" somit auf die angegebenen Werte.


```
[{"id":"2","name":"Blubb","autor":"bla","jahr":"2000","seiten":"100"}]
```

EKÜ:

- Der Webserver unterscheidet zwischen add, delete, update und read nicht anhand der URL, sondern anhand der verwendeten HTTP-Methode
  - Ein HTTP GET soll daher nur auslesen, ein HTTP DELETE soll löschen, ein HTTP POST soll hinzufügen und ein HTTP PUT soll updaten
  - Testen kannst du deinen Server z.B. mit der App Postman

EKV:

- Der Webserver persistiert (speichert) die Bücherliste in einem lokalen JSON-File
  - Beim Starten des Servers wird das File eingelesen
  - Bei jeder Änderung wird das File neu geschrieben
  - Verwende hierfür die "fs" Library von NodeJS

Abgabe

Ladet eure Abgabe hier mit dem Namen "*klasse_nachname*.js" hoch und kommt zu einem **Abgabegespräch**. Es ist **kein** Protokoll nötig.

Edit submission | Remove submission

Submission status

Submission status	Submitted for grading

Grading status	Graded
Last modified	Wednesday, 15 February 2023, 3:42 PM
File submissions	3cHIT_bohaczyk.js 15 February 2023, 3:42 PM
Submission comments	Comments (0)

Feedback

Grade	EK überwiegend
Graded on	Wednesday, 22 February 2023, 8:57 AM
Graded by	DD Dolezal Dominik