Tutorial – node.js

Inhalt

[Shell-Commands 1](#_Toc434785978)

[Installation von node.js 1](#_Toc434785979)

[Anlegen eines node.js-Projektes 1](#_Toc434785980)

[Ausführen eines node.js-Scripts 2](#_Toc434785981)

[Hinweis zu node.js mit GitHub 2](#_Toc434785982)

[Node.js Commands 2](#_Toc434785983)

[Beispielserver mit dem http-Modul 2](#_Toc434785984)

[Beispielserver mit dem Express-Framework 2](#_Toc434785985)

[Kommunikation mit dem Client über Socket.IO 3](#_Toc434785986)

[Serverseitig 3](#_Toc434785987)

[Clientseitig 3](#_Toc434785988)

[Aufgabe „Chat-Applikation“ 4](#_Toc434785989)

# Shell-Commands

|  |  |
| --- | --- |
| **node** | Öffnet die Node.js-Umgebung |
| **node** script.js | Führt script.js aus |
| **npm init** | Initialisiert ein neues Projekt |
| **npm install** | Installiert alle Abhängigkeiten eines Projekts |
| **npm install** module | Installiert ein bestimmtes Modul |
| **npm install** module **--save** | Installiert ein bestimmtes Modul und fügt es als Abhängigkeit in der package.json hinzu |
| **npm install** module **-g** | Installiert ein bestimmtes Modul global und macht es damit allen Projekten verfügbar |
| **npm test** | Führt das in der package.json aufgeführte Testscript aus |
| **npm start** | Führt das in der package.json aufgeführte Startscript aus |

# Installation von node.js

Node.js kann unter <https://nodejs.org/en/> heruntergeladen und installiert werden. Ein Windows Installer steht zur Verfügung. Nach einem Neustart des System kann über die Kommandozeile auf die globalen Pfadvariablen node und npm zugegriffen werden.

# Anlegen eines node.js-Projektes

1. Lege einen Ordner mit dem Namen deines Projektes an.
2. Öffne die Kommandozeile im Pfad deines angelegten Ordners.
3. Tippe npm init und folge den Instruktionen.
4. Lege ein node.js-Script an (eine .js-Datei).
5. *Optional*: Lege in der automatisch erzeugten package.json unter scripts ein Attribut „start“ zum Beispiel mit „node [your-script.js]“ als Wert an.
6. *Optional*: Installiere benötigte Module mit npm install [module] --save

Express: npm install express --save

Socket.io: npm install socket.io –save

# Ausführen eines node.js-Scripts

* Führe den Befehl node [your-script.js] in der Kommandozeile aus, um ein node.js-Script zu starten.
* Im Falle dafür, dass ein Startscript in der package.json festgelegt wurde, lässt sich das Projekt auch mit npm start ausführen.
* Hinweis: Man muss dafür im Ordner des node.js-Projektes sein!

# Hinweis zu node.js mit GitHub

Werden node.js-Projekte, zum Beispiel von GitHub, heruntergeladen, müssen mit npm install zumeist die Abhängigkeiten installiert werden, bevor das Projekt auch lokal lauffähig ist. Es ist nicht üblich die fremden node-Module zu pushen. Deswegen sollte der node\_modules-Ordner auch bei eigenen Projekten in der .gitignore gelistet werden.

# Node.js Commands

|  |  |
| --- | --- |
| **console.log("…");** | Erzeugt eine Ausgabe auf der Konsole des node.js-Servers |
| **require("…");** | Lädt ein bestimmtes Modul |

## Beispielserver mit dem http-Modul

Im folgenden Codebeispiel wird ein sehr einfacher Server aufgesetzt, der „Hello World“ im Browser ausgeben soll. Der folgende Server hört auf Port 1337.

|  |
| --- |
| var http = require('http');  http.createServer(function (*req*, *res*) {  *res*.writeHead(200, {'Content-Type': 'text/plain'});  *res*.end('Hello World\n'); }).listen(1337, "127.0.0.1", function() {  *console*.log('Server running at http://127.0.0.1:1337/'); }); |

Die Dokumentation zum HTTP-Modul von node.js findet sich unter folgendem Link: <https://nodejs.org/api/http.html>

## Beispielserver mit dem Express-Framework

Im folgenden Codebeispiel wird ein Beispielserver mit Express aufgesetzt, welcher statische Dateien aus einem Verzeichnis “public” ausliefert und zwischen den Routen /hallo und /about unterscheiden kann. Vorausgesetzt wird die Installation von express (siehe auch Anlegen eines node.js-Projektes).

var *express* = require('express');  
var *app* = *express*();  
  
*app*.get('/about', function (*req*, *res*) {  
 *res*.send('about');  
});  
  
*app*.get('/hallo', function(*req*,*res*) {  
 *res*.send('hallo.');  
});  
  
*app*.use(*express*.static('public'));  
  
var *server* = *app*.listen(3000, function () {  
 *console*.log('Example app listening at http://localhost:3000');  
});

Mehr Informationen zum:

* Routing: <http://expressjs.com/guide/routing.html>
* Ausliefern von statischen Dateien: <http://expressjs.com/starter/static-files.html>
* Aufsetzen eines Hello-World-Servers: <http://expressjs.com/starter/hello-world.html>

## Kommunikation mit dem Client über Socket.IO

Vorausgesetzt wird die Installation von Socket.IO (siehe auch Anlegen eines node.js-Projektes).

### Serverseitig

|  |  |
| --- | --- |
| Einbinden von socket.io | var *io* = require('socket.io')(*server*); |
| Erstmalige Verbindung eines Clients abfangen | *io*.on('connection', function (*socket*) {  // ... }); |
| Events abfangen *(im ersten Parameter des Callbacks befinden sich die mitgesendeten Daten)* | *socket*.on('your event name', function (*data*) {  *// ...* });  **Spezielles Event:**  *socket*.on('disconnect', function (*data*) {  *// ...* }); |
| Event an **alle** verbunden Clients auslösen *('your-data' sind die Daten, die mitgesendet werden sollen)* | io.emit('your event name', 'your data'); |
| Event für nur einen verbundenen Client auslösen *('your-data' sind die Daten, die mitgesendet werden sollen)* | socket.emit('your event name', 'your data'); |

### Clientseitig

|  |  |
| --- | --- |
| Einbinden von socket.io im head-Bereich des HTML-Dokumentes | <script src="https://cdn.socket.io/socket.io-1.3.7.js"></script> |
| Aufbauen einer Verbindung | var *socket* = io('http://localhost:3000'); |
| Events abfangen *(im ersten Parameter des Callbacks befinden sich die mitgesendeten Daten)* | *socket*.on('your event name', function (*data*) {  *// ...* }); |
| Event für den Server auslösen *('your-data' sind die Daten, die mitgesendet werden sollen)* | socket.emit('your event name', 'your data'); |

Weitere Informationen zu Socket.IO finden sich in der Dokumentation: <http://socket.io/docs/>

# Aufgabe „Chat-Applikation“

1. Anlegen eines Projekts namens „node-demo-chat“ und das Installieren von Express und Socket.IO. [siehe Anlegen eines node.js-Projekts]
2. Entsprechende Ordner (public-Folder) und Dateien (index.js und index.html) anlegen. [siehe Screenshot rechts]
3. Express-Server in der index.js aufsetzen. [siehe Beispielserver mit dem Expressframework]

var *express* = require('express'); // Express-Modul laden  
var *app* = *express*(); // Express-App erzeugen  
  
*app*.use(*express*.static('public')); // Statische Dateien aus dem Ordner public ausliefern  
  
var *server* = *app*.listen(3000, function () {  
 console.log('Chat app listening at http://localhost:3000');  
}); // Server auf Port 3000 hören lassen

1. Socket.IO-Server in der index.js aufsetzen. [siehe auch Kommunikation mit dem Client über Sockat.IO].

var io = require('socket.io')(*server*); *io*.on('connection', function (*socket*) {  
 *socket*.username = "Guest" + Math.round(Math.random()\*5000); // Zufälligen Usernamen erzeugen  
 *io*.emit('message', '<strong>' + *socket*.username + '</strong> connected'); // An alle: Benutzer ist verbunden  
 *socket*.on('message', function (*data*) { // Benutzer hat eine Nachricht gesendet  
 *io*.emit('message', '<strong>' + *socket*.username + '</strong>' + ': ' + *data*); // An alle: Nachricht von Benutzer  
 });  
 *socket*.on('disconnect',function() { // Benutzer hat den Chat verlassen  
 *io*.emit('message', '<strong>' + *socket*.username + '</strong> disconnected'); // An alle: Benutzer ist nicht mehr verbunden  
 });  
});

1. In der index.html ein HTML-Grundgerüst anlegen, welches im Body ein Inputfeld enthält, um Nachrichten zu schreiben und ein div, welcher später unser Container für die angezeigten Nachrichten wird:

<!doctype html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>**Node.js Demo**</**title**>  
</**head**>  
<**body**>  
 <**input** type=**"text"** id=**"chattext"** />  
 <**div** id=**"chathistory"**>  
 </**div**>  
</**body**>  
</**html**>

1. Socket.IO-Script einbinden . [siehe auch Kommunikation mit dem Client über Sockat.IO]

<**script** src=**"https://cdn.socket.io/socket.io-1.3.7.js"**></**script**>

1. Verbindung mit dem Server herstellen . [siehe auch Kommunikation mit dem Client über Sockat.IO]

<**script**>  
 var *socket* = *io*('http://localhost:3000');  
</**script**>

1. Auf Nachrichten reagieren: Die Nachricht wird dem div-Container hinzugefügt (Hinweis: Hier wird auch optional die Zeit angefügt und formatiert). [siehe auch Kommunikation mit dem Client über Sockat.IO]

*socket*.on('message', function (*data*) {  
 /\* Zeit formantieren \*/  
 var a = new Date();  
 var time =  
 ("0"+a.getHours()).slice(-2)+':'+("0"+a.getMinutes()).slice(-2)+':'+("0"+a.getSeconds()).slice(-2);  
  
 /\* Nachricht anfügen \*/  
 var message =  
 document.querySelector("#chathistory").innerHTML = document.querySelector("#chathistory").innerHTML +  
 "<br />[" + time + "] " + *data*;  
});

1. Nachrichten senden: Es wird ein Eventlistener zum Inputfeld hinzugefügt, welches auf ein Enter reagiert (alternativ hätte man auch einen Button hinzufügen können), es wird ein message-Event ausgelöst und das Inputfeld geleert. . [siehe auch Kommunikation mit dem Client über Sockat.IO]

window.onload = function() {  
 document.querySelector("#chattext").addEventListener('keypress', function (*e*) {  
 var key = *e*.which || *e*.keyCode;  
 if (key === 13) { // 13 is enter  
 *socket*.emit('message',document.querySelector("#chattext").value);  
 document.querySelector("#chattext").value = '';  
 }  
 });  
};

1. Wir sind nun fertig. Die Chat-Applikation kann mit node index.js gestartet werden. [siehe Ausführen eines node.js-Scripts] und über <http://localhost:3000> im Browser erreicht werden. Getestet werden kann die Applikation, indem man mehrere Browsertabs verwendet.

