Node.js

Tutvustus

# Sissejuhatus

Node.js on tarkvaraplatvorm, mis võimaldab programmeerida serveripoolseid rakendusi kasutades JavaScript’i. Node-I siseselt kasutab see Google’I V8 mootorit koodi käivitamiseks ning sisaldab HTTP server teeki, võimaldades jooksutada veebiserverit ilma Apache või muu veebiserverita.

Node loodi aastal 2009 Ryan Dahl-I poolt, aastal 2011 toodi välja ka NPM (Node Package Manager), mis võimaldab lihtsalt hallata mooduleid.

# Hello, World

Node.js-i kasutamiseks on mitu võimalust. Minnes node-I kodulehele ([www.nodejs.org](http://www.nodejs.org)) saame edasi liikuda Downloads lehele, kust saame Node-i alla tõmmata. Mina käivitan Node-I 64-bitisel Linux-I serveril ja seetõttu valin ka Linux Binaries (.tar.gz) alt 64-bit faili.





Selle õpetuse raames käivitame node-I viidates manuaalselt selle asukohale, kuid on ka võimalik süsteemi paigaldada nii, et selle binaarfail on kättesaadav lihtsalt trükkides *node.* Mina aga kasutades wget-i, tõmban selle oma kohalikku kausta ja pakin lahti. Seejärel kustutan allatõmmatud tar faili ning nimetan lahti pakitud kausta lühemaks.

wget <http://nodejs.org/dist/v0.10.26/node-v0.10.26-linux-x64.tar.gz>  
tar -xzf node-v0.10.26-linux-x64.tar.gz  
rm node-v0.10.26-linux-x64.tar.gz  
mv node-v0.10.26-linux-x64 node

Edaspidi saan käivitada node-i REPL-ina (Read-Eval-Print-Loop), mis on olemuselt sarnane veebilehitsejates leiduvate konsoolidega. Käivitamiseks pean viitama node-i binaarfailile kasutades selle asukohta.

./node/bin/node

REPL-is saan trükkida JavaScript-i käske ja kõige elementaarsem oleks proovida konsooli väljastada rea Hello, World. Selleks trükingi JavaScript-i käsu:

console.log("Hello, World");

Rakendusest saan väljuda vajutades ctrl+C kaks korda.

# Veebiserveri püstipanemine

Järgmisena üritame panna püsti kõige elementaarsema veebiserveri, mis vastab päringule läbi veebilehitseja lihtsalt tekstiga Hello, world.

Esmalt loon uue kausta, kuhu sisse paigutan meie faili esimene.js. Selle sisu on järgnev:

var http = require("http");

http.createServer(function(req, res){

res.writeHead(200, {"Content-Type": "text/plain"});

res.end("Hello, World");

}).listen(8000);

console.log("Server jookseb pordil 8000");

Esmalt laeme sisse mooduli *http*, mis on node-ile sisseehitatud. Seejärel kasutame http meetodit *createServer*, argumendiks anname funktsiooni, mida käivitada siis, kui keegi läheb meie lehele. Sellele funktsioonile antakse kaks parameetrit, *request* ja *response* (päring ja vastus). Vastusele anname staatuskoodi 200 (OK) ning märgime ära, et vastus on tavalises tekstiformaadis. Lõpetame sellega, et kirjutame kasutajale Hello, World.

Kui minna brauseris aadressile serveriaadress:8000 meile see ka väljastatakse.

Sellega on kõige lihtsam veebiserver tehtud. Edasi vaatame failide väljajagamist, mida teeme juba väliste moodulitega.

# Failide väljajagamine

Kuna failide väljajagamiseks kasutame ühte välist moodulit, siis siin peatükis vaatame ühtlasi ka NPM-i (Node Package Manager-i). Teen kausta nimega teine ja käivitan npm-i parameetriga init:

../node/bin/npm init

Seda tehes käivitub package.json faili tegemine, milles saab määrata ära, mis versiooniga on tegemist, mis mooduleid meie projekt vajab, kes on autor jne.

name: (teine) teine

version: (0.0.0)

description: Meie teine projekt

entry point: (index.js)

test command:

git repository:

keywords:

author: Kristjan

license: (ISC)

About to write to /home/kristjan/nodeope/teine/package.json:

{

"name": "teine",

"version": "0.0.0",

"description": "Meie teine projekt",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Kristjan",

"license": "ISC"

}

Is this ok? (yes) yes

Versiooninumber peab olema formaadis \*.\*.\*, muidu package.json faili ei looda. Nüüd peaks meil olemas olema meie package.json fail ja saame sinna lisada, mis mooduleid me saame kasutada. Avangi package.json koodiredaktoris ning muudan selle sisu:



Lisades omaduse dependencies saan määrata ära, mis mooduleid meie rakendus vajab, me kasutame moodulit *express*, tärniga (“\*”) märgime ära, et vajame kõige uuemat versiooni sellest. Saaksime ka kasutada süntaksit nagu >= 1.2.0 et märkida, et soovime versiooni, mis on vähemalt 1.2.0 või uuem või muid analoogseid märgendeid.

Moodulite installimiseks kasutame taaskord npm-I, kuid seekord koos parameetriga install

../node/bin/npm install

Kui kõik on õigesti tehtud, siis peaks npm paigaldama ära kõik moodulid, mis meil *dependencies* atribuudi all märgitud oli, ning tegema kokkuvõtte sellest kõige lõpus (lisaks paigaldatakse kõik moodulid, mida need kasutavad).

Nüüd saame luau oma faili, package.json-is märkisime selle nimeks index.js.

var express = require("express"),

app = express();

app.get("/", function(req, res){

res.send("Hello, World");

});

app.get("/tere", function(req, res){

res.send("Tere, Maailm");

});

app.listen(8000);

Siin juhul mines aadressile serveriaadress:8000 väljastatakse meile Hello, World, mines aga lehele serveriaadress:8000/tere, väljastatakse Tere, Maailm.

Expressi kasutades saame aga väga lihtsalt öelda, et hakaku mingit kausta jagama. Teen käesolevasse kausta omakorda kausta avalik ja panen sinna lihtsa html faili sisse (index.html):

<!DOCTYPE html>

<html lang="en">

<head>

<title>Hello, World</title>

</head>

<body>

<h1>Hello, world</h1>

<body>

</html>

Seejärel muudan oma index.js-i sisu:

var express = require("express"),

app = express();

app.use(express.static(\_\_dirname + "/avalik"));

app.listen(8000);

Nüüd käivitades node-i index.js-iga saan näha, et jagataksegi meile välja meie loodud html fail-i

# WebSocket

Tähtis osa Node.js-i juures on see, kui lihtne on WebSocket-ite kasutamine socket.io mooduliga. Loon uue projekti kausta kolmas, kus teen jälle läbi npm init-i. Seekord panen package.json faili lisaks express-ile ka socket.io:

{

"name": "kolmas",

"version": "0.0.0",

"description": "Katsetame WebSocketeid",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"repository": "",

"author": "Kristjan",

"license": "BSD",

"dependencies": {

"express": "\*",

"socket.io": "\*"

}

}

Jooksutades taaskord npm install-I, paigaldatakse meile need moodulid ning saame jätkata. Sarnaselt eelmisele peatükile, loon faili index.js ning kausta avalik, mida hakkan välja jagama.

var express = require("express"),  
 app = express(),  
 server = require("http").createServer(app),  
 io = require("socket.io").listen(server);  
  
app.use(express.static(\_\_dirname + "/avalik"));  
server.listen(8000);  
  
io.sockets.on("connection", function(socket){  
 socket.emit("edasi", {tekst: "Hello, World"});  
  
 socket.on("tagasi", function(andmed){  
 console.log("Tagasi saadeti: " + andmed.tekst);  
 });  
});

Siin loome server kasutades nii express-I kui ka http moodulit. Muutuja io hakkab sisaldama socket.io moodulit, ning paneme selle kuulama samat porti, mis meie server.

Märgime ära, et see kasutaks meie kausta avalik ning pordiks jällegi 8000. Edasi määrame, et kui keegi ühendab (io.sockets.on("connection" … ) siis käivitame selle funktsiooni. Funktsioonile antakse kätte objekt socket, millele saame lisada sündmuste kuulareid ning ühtlasi ka saame saata andmeid. Siin me teemegi nii, et kõigepealt saadame *emit* meetodiga sellele ühendusele objekti (me ei pea saatma objekti, võime ka saata *string*, *number*, *array* ja muid andmetüüpe). Sõnumi nimeks paneme edasi. Ühtlasi lisame sellele kuulari, kasutades meetodit *on.* Hakkame kuulama sõnumit nimega tagasi, ning sel juhul käivitame funktsiooni, mis saab vastu andmed.

Kliendi poolel on WebSocket-ite kasutamine üsnagi sarnane. (index.html)

<!DOCTYPE html>

<html lang="et">

<head>

<script src="/socket.io/socket.io.js"></script>

<script>

var socket = io.connect("http://serveriaadress");

socket.on("edasi", function(andmed){

console.log(andmed);

socket.emit("tagasi", {tekst: "Tere, Node"});

});

</script>

</head>

</html>

Socket.io teek jagab automaatselt faili */socket.io/socket.io.js*, mis lihtsustab oluliselt ka kliendipoolset WebSocket-ite kasutamist. Edasi nõuame *socket* objekti ja lisame sarnaselt serverile sellele kuulari. Sõnumi nimeks paneme edasi, ning kui selline sõnum tuleb, käivitame funktsiooni, mis saab kätte selle sõnumi sisu. Kuvame selle sõnumi sisu konsooli ning saadame vastuse.

# Klientidevaheline suhtlus

Nüüd, kus oleme tutvust teinud WebSocket-itega, üritame luua rakenduse, mis edastab ühelt lehekülje külastajalt sõnumeid teisele. Esmalt muudan oma index.js faili, et see edastaks sõnumeid selle asemel, et neid konsooli kuvada. Saaksime kasutada käsku io.sockets.emit ... kuid see saadaks kõikidele, kaasaarvatud saatjale sõnumi. Selle asemel kasutame socket.broadcast.emit meetodit, mis saadab kõikidele, peale iseenda. (index.js)

var express = require("express"),  
 app = express(),  
 server = require("http").createServer(app),  
 io = require("socket.io").listen(server);  
  
app.use(express.static(\_\_dirname + "/avalik"));  
  
server.listen(8000);  
  
io.sockets.on("connection", function(socket){  
  
 socket.on("sonum", function(andmed){  
 socket.broadcast.emit("sonum", {  
 sonum: andmed,  
 autor: socket.kasutaja  
 });  
 });  
  
 socket.on("minuNimi", function(andmed){  
 socket.broadcast.emit("uusKasutaja", andmed);  
 socket.kasutaja = andmed;  
 });  
  
 socket.emit("kuidasNimi", null);  
  
});

Kasutajalt kõigepealt küsitakse tema nime. Kui kasutaja vastab, siis pannakse see nimi *socket* objektile atribuudina külge ja öeldakse kõigile teistele, et liitus uus kasutaja. Nüüd jääb üle vaid muuta meie veebilehte nii, et lehele minnes küsitakse kasutajalt nime, ning seejärel saab ta kasti kirjutada ja teistele sõnumeid saata. (index.html)

<!DOCTYPE html>  
<html lang="et">  
 <head>  
 <script src="/socket.io/socket.io.js"></script>  
 <script>  
 var nimi = window.prompt("Mis nimeks?");  
 var socket = io.connect("http://tammekivi.ee");  
 socket.on("sonum", function(andmed){  
 document.getElementById("sonumid").innerHTML += '<p><b>' + andmed.autor +  
 ': </b>' + andmed.sonum + '</p>';  
 });  
  
 socket.on("kuidasNimi", function(){  
 socket.emit("minuNimi", nimi);  
 });  
  
 socket.on("uusKasutaja", function(andmed){  
 document.getElementById("sonumid").innerHTML +='<p><i>Liitus uus kasutaja: '+  
 andmed +'</i></p>';  
 });  
  
 function saadaSonum(){  
 socket.emit("sonum", document.getElementById("minusonum").value);  
 document.getElementById("sonumid").innerHTML += '<p><b style="color:red;">' +   
 nimi +': </b>' + document.getElementById("minusonum").value + '</p>';  
 document.getElementById("minusonum").value = "";  
 }  
 </script>  
 </head>  
 <body>  
 <input type="text" id="minusonum"><input type="submit" value="Saada"  
 onclick="saadaSonum()">  
 <div id="sonumid"></div>  
 </body>  
</html>

Edasised ülesanded

* Võtta kasutusele socket-ite toad (room - <https://github.com/LearnBoost/socket.io/wiki/Rooms>)
* Näidata klientidele, kes hetkel on toas (saame kuulata sündmust disconnect, et teada, kui keegi lahkub)
* Luua administraatori süsteem, kes saab näiteks välja visata või lahti ühendada teisi socketeid (socket.disconnect() meetod)

# Lingid

<http://nodejs.org/> – Node.js koduleht  
<http://nodejs.org/api/> - Node.js dokumentatsioon  
<http://expressjs.com/> - Express raamistiku koduleht  
<http://socket.io/> - socket.io koduleht  
<https://www.npmjs.org/> - NPM koduleht