Laboratorul 02: Introducere in NodeJS si Express

Ce este NodeJS?

Prezentare pe scurt

NodeJS [https://nodejs.org/en/] este un ecosistem de Javascript care permite folosirea limbajului Javascript pe partea de server. Este bazat pe motorul V8 de la Chrome, ruleaza pe un singur fir de executie si este asincron, utilizand o bucla de evenimente pentru a executa operatiile cat mai rapid.

Structura unui proiect in NodeJS

Module

Un proiect de Node este structurat in **module**. Fiecare fisier Javascript din componenta unui proiect este considerat un modul. Puteti sa faceti analogia cu fisierele **.c** din C sau cu **clasele** din Java.

Un modul poate fi importat de catre oricare alt modul (adica fisier .js), folosind keyword-ul **require** si calea relativa catre el. Un modul poate expune informatii in exterior folosind proprietatea **exports** a obiectului global **module**.

Se pot exporta oricate variabile sau functii. Chiar daca se exporta doar o functie, ca in exemplul urmator, este indicat sa o puneti intr-un obiect atunci cand o exportati

```
const sayHello = () => {  console.log("Hello world!");  }
module.exports = {
  sayHello
}
/*
module.exports = { sayHello } e identic cu

const objForExport = { sayHello: sayHello };
module.exports = objForExport;
*/
```

Atunci cand creati un obiect, daca cheia si valoarea au aceeasi denumire, se poate scrie doar numele cheii, fara sa se mai puna **:** si valoarea

modul1.js

```
const a = 2;
const b = 3;
const getA = () => { return a; }
const getB = () => { return b; }
module.exports = {
  getA
};
// pentru oricine va importa acest modul, singurul lucru la care va avea acces va fi functia "getA"
```

modul2.js

```
const myModule = require('./modul1.js');

const a_from_module_1 = myModule.getA();
console.log(a_from_module_1); //2

const b_from_module_1 = myModule.getB(); //ReferenceError!
```

Pachete

Asa cum in alte limbaje se pot folosi biblioteci externe pentru diverse functionalitati ce sunt deja implementate (exemplu, stdio.h in C pentru I/O) si in Node se pot folosi **pachete**.

Pachetele sunt module instalate din surse externe. Pentru a instala un pachet se foloseste NPM - node package manager [https://www.npmjs.com/] in terminal.

Pentru a putea instala pachete intr-un proiect, este nevoie ca aceasta sa fie initializat. Pentru a initializa un proiect de node, este nevoie sa rulati comanda

```
npm init
```

In urma rularii acestei comenzi se va crea un fisier, **package.json**. Acest fisier are rolul de a retine informatii cu privire la proiect si la dependentele sale.

Daca vreti sa rulati proiectul pe o alta masina, este nevoie sa va instalati local toate pachetele pe care le folositi in proiect. Puteti sa le instalati pe fiecare de mana, sau puteti sa va folositi de package.json si ele vor fi instalate automat

Pe langa dependente, in **package.json** sunt retinute si informatii cu privire la proiect (autor, repository, descriere) precum si scripturi de rulare si testare. Puteti asocia partea de **scripts** din package.json cu un Makefile.

Dupa ce ati initializat proiectul, puteti incepe sa instalati pachete. Pentru a instala un pachet, se executa comanda urmatoare:

```
npm install nume_pachet --save
```

Dupa ce instalati un pachet, se vor crea folderul **node_modules** si fisierul **package-lock.json**. In folder se afla toate pachetele instalate. In fisier se retine versiunea exacta a pachetului cu care se lucreaza.

Niciodata sa nu stergeti package-lock.json. De asemenea, acesta trebuie obligatoriu inclus in git.

Pentru a utiliza un pachet instalat, se foloseste tot keyword-ul **require**, insa se scrie doar numele pachetului, nu si calea catre locul unde a fost descarcat, aceasta deducandu-se automat din **node_modules**.

```
//presupunem ca am instalat inainte pachetul fictiv my-awesome-package

const myAwesomePackage = require('my-awesome-package');
//do stuff
```

Pentru a vedea cum se utilizeaza un pachet, este **necesar** sa cititi pagina acestuia de pe NPM sau de pe site-ul pachetului, daca exista. De obieci, documentatiile sunt cuprinzatoare.

Rest API in NodeJS

Framework propus

NodeJS este perfect pentru API-uri de tip REST. Acesta ofera posibilitati out of the box pentru acest lucru, insa noi va recomandam sa utilizati ExpressJS [https://expressjs.com/], cel mai popular pachet de pe NPM si de asemenea cel mai popular framework pentru NodeJS.

Express ofera posibilitatea de a crea rute HTTP foarte usor. De asemenea, parsarea continutului din cadrul cererilor HTTP este realizata mult mai simplu in express, decat in alte frameworkuri. Avantajul major se resimte in natura sa minimala si nerestrictiva.

Structura API

Exista doua moduri principale in care va puteti organiza codul atunci cand doriti sa realizati un API de tip REST. Puteti sa va impartiti proiectul in:

- foldere care inglobeaza **scripturi care se afla pe acelasi nivel logic** in aplicatie (e.g. folder dedicat rutelor, folder dedicat interactiunii cu baza de date, etc…)
- foldere dedicate **functionalitatilor individuale** (e.g. folder pentru utilizatori, folder pentru comenzi, etc…).

Chiar daca structura codului este importanta, cel mai important factor intr-un API este modularitatea la nivel de functionalitati. Cu cat fiecare functionalitate are un context bine definit, cu atat este mai putin probabil sa intampinati erori de logica sau de cod.

Rularea unui API

Pentru a rula un API de Node, este nevoie sa definiti un fisier de start. Acest fisier trebuie sa includa apoi referinte catre celelalte fisiere ce fac parte din proiect, atat direct, cat si indirect (e.g. start face referire catre modulul 1 iar modulul 1 face referire catre modulul 2).

Exemplu de API minimal cu express:

start.js

```
const express = require('express');

const app = express();

app.get('/', (req, res) => {
    res.send("Hello world!");
});

app.listen(3000);
```

Daca doriti sa utilizati scripturi de **npm**, va trebui sa includeti referinta catre fisierul de start si in **package.json**:

package.json

```
{
  "name": "lab2",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node start.js"
  },
  "dependencies": {
    "express": "~4.16.1"
  }
}
```

Pentru a rula serverul, executati comanda:

```
npm run start
```

Atunci cand dezvoltati, va recomandam sa folositi pachetul Nodemon [https://www.npmjs.com/package/nodemon]. Acesta urmareste modificari in cod si reporneste automat serverul. In cazul in care vreti sa folositi nodemon, va trebui sa inlocuiti **node** cu **nodemon** in **package.json**.

Testarea unui API

Pentru a testa un API scris in Node, puteti sa folositi **cURL**, **wget** sau Postman [https://www.postman.com/]. Noi va recomandam Postman, pentru ca ofera multe functionalitati utile, precum variabile de mediu, colectii, scripturi automate, etc…

Exercitii

1. Creati 3 module de Node.
    a. Primul modul trebuie sa expuna o functie care calculeaza suma elementelor dintr-un vector dat ca parametru si returneaza suma. Suma trebuie calculata folosind functia **reduce**.
    b. Al doilea modul trebuie sa expuna o functie care primeste ca parametru un vector **vec** si un numar **parNum**. In aceasta functie se va apela functia din modulul anterior, dandu-se ca parametru un vector nou format din elementele din **vec** care au aceeasi paritate cu **parNum**. Trebuie sa folositi **filter** pentru generarea vectorului nou. Se va returna suma.
    c. Al treilea modul va folosi functia din al doilea modul, dandu-i ca parametru un vector, un numar si va afisa rezultatul intors.
2. Creati un proiect de NodeJS gol.
3. Instalati pachetul **moment** si afisati ora curenta in formatul YYYY-LL-ZZThh:mm.
4. Instalati **ExpressJS** si rulati scriptul din laborator.
5. Inlocuiti “Hello world” cu ora curenta obtinuta la exercitiul 3.
6. Folositi Postman pentru a testa ruta.

NU uitati sa puneti fisierul **.gitignore** si sa adaugati in el **node_modules**. Solutiile care nu contin **.gitignore** nu vor fi luate in considerare

pw/laboratoare/02.txt · Last modified: 2021/03/17 08:06 by alexandru.hogea