NODE JS

Skripta

Sadrzaj

[1. Uvod u Node.js 3](#_Toc105444253)

[2. Moduli 4](#_Toc105444254)

[2.1. Core moduli 5](#_Toc105444255)

[3. NPM 6](#_Toc105444256)

[4. Setup 8](#_Toc105444257)

[Nodemon 8](#_Toc105444258)

[Express 8](#_Toc105444259)

[Mongoose 8](#_Toc105444260)

[CORS-cross origin resource sharing 8](#_Toc105444261)

[Parsovanje 9](#_Toc105444262)

[Server setup 9](#_Toc105444263)

[DataBase setup 10](#_Toc105444264)

# Uvod u Node.js

Node.js nam omogucuje da programiramo sam racunar i da koristimo JavaScript van borwsera. JavaScript nema mogucnost da direktno programira racunar, dok C++ ima. V8 Engine je kompajler za JavaScript. V8 Engine je programiran u C++ i zbog toga preko Node.js mozemo da kazemo C++ kako zelimo da programiramo sam racunar.

Uz pomoc Node.js mozemo da kontrolisemo servere, da upravljamo bazama podataka, da kreiramo Web app i Desktop App.

**Da bi mogli da koristimo Node.js, potrebno je da skinemo node.js i instaliramo na nas racunar.**

**!!!U sklopu instalacije, automatski ce se instalirati i npm.!!!**

NPM(Node package manager) omogucuje da se lako upravljanje paketa modula unutar node.js. Koristi se unutar command line.

# Moduli

Za razliku od koriscenja script linkovanja unutar index.html fajla, gde unutar njega dobijamo kopije svih tih .js fajlova koje smo linkovali, u node.js koristimo module.

Moduli nam omogucavaju da razmenjujemo podatke izmedju razlicitih .js fajlova.  
Ovaj princip moze se videte u PHP-u gde uz pomoc require dobijamo kopiju drugog fajla unutar trenutnog fajla.

Svaki modul unutar node.js na kraju koda ima ugradjen module.exports.

module.exports **predstavlja prazan objekat** unutar kojeg smestamo sve ono sto zelimo da delimo sa ostalim modulima.

Primer:

const ***UserModel*** = ***mongoose***.model('users', userSchema);  
  
module.exports = ***UserModel***;

**!!! Ovde nismo dodali exports objektu novi property, vec smo ga prepravili da on ne bude vise prazan objekat vec UserModel.** Sada UserModel mozemo da koristimo u drugim modulima:

const ***Users*** = require('./models/userModel');

Kao sto vidimo, koristimo kljucnu rec require i navodimo putanju do fajla u kojem se nalazi UserModel.

**!!! Putanju je potrebno navoditi samo za custom module, koji nisu ugradjeni ili instalirani unutar node.js projekta.**

## Core moduli

U Node.js postoje vec predefinisani moduli koji se mogu koristiti.

Neki od njih su:

FS

Events

Http

NET

OS

PATH

READLINE

URL

ZLIB

QUERYSTRING

Ove module zahtevamo bez navodjenja apsolute ili relativne putanje.

const fs=require(“fs”);

# NPM

NPM(Node package manager) omogucuje da se lako upravljanje paketa modula unutar node.js. Koristi se unutar command line. Automatski se instalira kada pokrenemo isntalaciju node.js.

Uz pomoc npm mozemo lako da instaliramo dodatne module unutar naseg projekta.

Pre toga, potrebno je da inicijalizujemo nas projekat.

Uz pomoc command line udjemo u nas folder gde planiramo da se nalazi nas projekat i pokrenemo komandu:

npm init

U nasem folderu, kreirace se package.json fajl sa osnovnim podacima o nasem projektu.

Primer:

{  
 "name": "backend",  
 "version": "1.0.0",  
 "description": "Online shop",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "author": "We",  
 "license": "ISC",  
   
}

Nakon toga, mozemo da isntaliramo dodatne module.

Uz pomoc komande:

npm i <naziv paketa> -S

-S navodimo, kako bi u package.json fajlu bilo upisano sta smo instalirali.

Primer:

npm i express -S

Ako nam je prvi modul koji instaliramo, u nasem root folderu ce se kreirati “node-modules” folder u kojem ce biti smesteni svi fajlovi koji su vezani za nas instaliran modul. Takodje, svi fajlovi za svaki sledeci instaliran modul bice smesteni u taj isti folder.

U “dependencies” bice upisano da smo instalirali express modul:

"dependencies": {  
 "express": "^4.18.1"

Upravo uz pomoc package.json fajla, ako smo upisivali svaki instaliran modul, mozemo da jednostavno instaliramo sve module u slucaju brisanja foldera “node-modules”.

Jednostavno pokrenemo komandu:

**npm i**

i svi moduli koji su upisani u dependencies bice instalirani ponovo.

# Setup

Da bi zapoceli projekat u Node.js, neophodno je da instaliramo nekoliko dodatnih modula koje cemo koristiti u projektu.

## Nodemon

Da ne bi manuelno restartovali server svaki put kada zelimo nesto da vidimo, koristicemo nodemon koji to automatski radi.

Instaliranje nodemon modula:

npm i nodemon -S

Pokretanje fajlova uz pomoc nodemona:

nodemon <naziv fajla>

## Express

Express je web aplikacioni framework koji sadrzi dodatke za rad sa web i mobilnim aplikacijama.

Instaliranje expressa:

npm i express -S

## Mongoose

Ovaj modul nam omogucava rad sa Mongo DB bazom podataka i kreiranje seme za podatke koje zelimo da smestimo u bazu podataka.

Instaliranje Mongoose:

npm i mongoose -S

## CORS-cross origin resource sharing

Ovaj modul koristimo da automatski omogucimo da backend prima requeste sa frontenda.

Instaliranje cors-a:

npm i cors -S

Setovanje u index.js:

app.use(cors());

## Parsovanje

Request sa frontenda se vraca u masinskom kodu. DA bi mogli to da iskorsitimo, potrebno je da ga parsiramo u JSON. To nam omogucava Body parser koji je ugradjen u express modul. Potrebno je samo da u index fajlu setujemo:

app.use(***express***.***urlencoded***({ extended: false }));//If extended is false, you can not post "nested object"  
app.use(***express***.***json***());//Frontend request convert in JSON

## Server setup

Pored toga, moramo da setujemo server:

const port = 4000;  
const ***ServerConfig*** = {  
 port: port,  
 serverRunningMsg: `Server is running on port: ${port}`  
};  
  
module.exports = ***ServerConfig***;

//SERVER LISTENING  
app.listen(***serverConfig***.port, err => {  
 if (err) {  
 ***console***.log(err);  
 }  
 else {  
 ***console***.log(***serverConfig***.serverRunningMsg);  
 }

## DataBase setup

Povezivanje sa bazom vrsimo preko moongose modula:

Kreiramo dbConfig.js fajl u koji smestamo podatke koji su nam potrebni za konekciju i setujemo mongoose options:

const MONGO\_PASS = 'novomak2020';  
const MONGODB\_URL = `mongodb+srv://novomak:${MONGO\_PASS}@cluster0.ivtbn.mongodb.net/?retryWrites=true&w=majority`;  
const mongooseOptions = {  
 useNewUrlParser: true,//Enable to se new Parser  
 useUnifiedTopology: true,  
 useCreateIndex: true,  
};  
  
module.exports = {  
 MONGO\_PASS: MONGO\_PASS,  
 MONGODB\_URL: MONGODB\_URL,  
 mongooseOptions: mongooseOptions  
};

Zatim u index.js fajlu vrsimo povezivanje

//CONNECT TO MONGO DB  
***mongoose***.connect(***dbConfig***.MONGODB\_URL)  
 .then(data => ***console***.log('MONGO DB is connected.'))  
 .catch(err => ***console***.log(`Error while connecting to MONGO DB: ${err}`));

Za razliku od SQL baze, Mongo DB koristi JSON, odnosno BSON (Binary JSON) kao format podataka.

Da bi osigurali jedistveni sablon(format podataka) kakav zelimo da cuvamo u bazi podataka, neophodno je da kreiramo semu, odnosno model podataka:

const ***mongoose*** = require('mongoose');  
  
const userSchema = new ***mongoose***.Schema({  
 username: {type: String, required: Boolean},  
 password: {type: String, required: Boolean},  
 email: {type: String, required: Boolean},  
 firstName: {type: String},  
 lastName: {type: String},  
 avatar: {type: String},  
 gender: {type: String},  
 address: {type: String},  
 city: {type: String},  
 postCode: {type: String},  
 phoneNumber: {type: Number},  
  
 isAdmin: {type: String, required: Boolean, default: false},  
});  
  
const ***UserModel*** = ***mongoose***.model('users', userSchema);  
  
module.exports = ***UserModel***;