Installem NodeJs de nodejs.org i desprem obrim la consola desde la terminal cmd.

Realitzem les seguents instruccions de comprobacio…

npm -v

node -v

Que és npm? Gestor de paquets, control de instalacions y actualitzacions.

Podem opcionalment realizar la instrucción npm install -g npm per actualitzar el gestor de paquets a la ultima versió.

Ara anem a instalar una base de dades que ens permetra treballar amb un sistema d’enmagatzematge de dades de forma perenne i perque no, generar una petita aplicación que podreu endurvos al finalizar la capsula.

Podem obtener la versió Community de <https://www.mongodb.com/try/download/community>

Que és Mongo Db?

Es una base de dades no relacional, significa que no tenim taules ni registres ni joins ni taules relacionades, a on treballem unicament amb objectes json o bson que és una representacio binaria de json. De les principals ventatges és la flexibilitat que ofereix les bd nosql que tenen absencia d’estructura i ens permeten definir una distribució lliure, flexible i escalable.

El seguent pas és crear una carpeta en c:/data i dins un altre anomenada db per les bases de dades i configuracions de Mongo

Despres ens dirigim a c:/Archivos de programa/Mongo Db/Server/5.0/bin i executem:

* Mongod.exe
* Mongo.exe és la terminal per gestionar la bd

A continuación anem a crear el diagrama per conceptualitzar la base de dades que definirem conjuntamente amb les seves entitats i relacions. En el nostre exemple, generarem una aplicación que vinculara imatges a usuaris.

USUARI

* \_id
* nom
* cognom
* email
* clau

IMATGES

* \_id
* idUser
* arxiu

Ara anem a crear un projecte de NodeJs i per aixó tenim que cercar un directori base a on desplegar el projecte.

Dirigimnos a Documents i allí crearem un directori anomenat NodeJs amb la instrucción mkdir NodeJs

A continuación anem a iniciar l’asistent per instal.lar el projecte i per aixó executarem npm init.

A continuación ens demanara el nom del paquet?

També pregunta la versió i podem deixar la 1.0.0 que és la suggerida.

Adicionalment demana la descripción del projecte, a on podem indicar “Exemple de api rest per el curs de introducción a nodejs.

El entry point el tenim que deixar en index.js, essent el punt inicial del nostre projecte.

A continuació les opcions “Test command”, “git repository” i “keywords” no les personalitzarem ja que són conceptes més avançats.

Després l’autor el definirem amb el nostre nom.

Per ultim definim el tipus de llicenciament si o desitgem.. podem posar MIT per exemple.

I ja només queda el confirmar la confirmació i instal.lació.

Un cop finalitzat podem executar un dir per veure el contingut que ha elaborat i podrem observar l’arxiu package.json a on dins és definirán el registre de configuracions i dependencies.

Per poder treballar be i aconse

Framework express

npm install express –save per que aquesta llibreria es guardi en les dependencias del package.json i aixi quan nosaltres traslladem el codi a un altyre lloc i executem npm install, installi totes les dependencias automaticament.

Body-parser

El que fara es parsear les peticions a format objecte json i aixi poder treballar amb ells correctamente.

npm install body-parser –save

Connect-multiparty

És una llibreria que permet pujar arxius el servidor amb l’ajuda de nodejs i el protocol http.

Npm install connect-multiparty –save

ORM mongoose

Npm install mongoose –save

Lllibreria per paginar els resultats i que treballa amb el ORM

Npm install Mongose-pagination –save

Realitzara la recarrega del servidor en cada modificación que actualitzem i evitant que tinguem que executar el servidor manualment

Npm install Nodemon –save-dev //només és desplegara en desenvolupament en local i en producción no

Anem a crear una base de dades ara amb la instruccio use curs\_ndejs o graficament

Ara tenim que indicar-li que tenim que treballar amb la bd correcponent switched to db curs\_nodejs

I definim un nou registre en una colleccio amb db.usuaris.save({nom: “Oriol”, cognom: “Tinoco”, email: “a@a.es”, clau: “123456”}); o graficament.

Per comprobar que s’ha enregistrat correctamente, hem d’executar db.users.find(); o graficament.

Conexio a la base de dades

Crearem un arxiu anomenat index.js i indicarem el següent:

‘use strict’ per poder indicar instruccions dels nous estándar de javascript

Ara anem a carregar el modul de mongodb através del orm mongoose que hem escollit com a intermediari

var mongoose= require(‘mongoose’);

mongoose.connect(‘mongodb://localhost:27017/curs\_nodejs’,(err, res) =>{

if(err){

throw err;

}else{

Console.log(“La base de dades está funcionant correctamente”);

}

});

Crear un servidor web per probar

Podriem realizar-ho amb nodemon amb nodemon i el nom del arxiu, pero podem aconseguir automatitzar aquest procés emprant una modificación en el package.json i que executara tot alhora al executar npm start.

En el área de scripts de package.json

“scripts”: {

“start”: “nodemon index.js”,

“test”: “echo \”Error: no test specified\” && exit 1”

},

Ara ja podem executar npm start i observar com contesta el nostre script, confirmant que s’ha establert la conexión a la base de dades correctamente.

Si realitzem una modificación sobre l’script, al guardar els canvis el nodemon actualitza automaticament l’execució del servidor.

Crearem ara un arxiu app.js

'use strict'

var express = require("express");

var bodyParser = require("body-parser");

var app = express();

*//Carregar  rutes*

app.use(bodyParser.urlencoded({extended:false})); *//Configurar  bodyparser*

app.use(bodyParser.json());

*//configurar capçaleres*

*//Rutes base*

module.exports = app;

I Després hem de carregar l’script de app.js desde el index.js, definir el port i la funció del servidor web.

Ja un cop executat, podem accedir a un navegador amb localhost:3977 i veure com respon.

Després podem fer un exemple de ruta accedint a app.js i realitzant el seguent codi:

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

    res.status(200).send({message: "Llistat d'usuaris"})

});

Ara ja estem preparats per realizar els nostres primers models, que son representacions de les entitats de les bases de dades i que ens permeten treballar amb elles de forma eficient.

Iniciarem creant una carpeta anomenada models i dins d’aquesta crearem el primer model anomenat usuaris.js

Recordem el emprar “user strict”, generem un Schema amb l’estructura dels camps definits en la bd i per ultim exportem el Schema amb un objecte anomenat Usuari. Important emprem el singular ja que en la colecció de MongoDb Mongoose ho pluralitza.

Consulta usuaris.js

Realitzem el altre model imatge.js, definint el camp arxiu i el camp usuari, a on definim la referencia a l’objecte usuari corresponent.

Consulta imatge.js

NOTA

Per aconseguir eliminar el avis de mongoose que retorna per la consola a on hem realitzat el npm start, tenim que afegir la següent linia de codi, just a sobre de la crida a mongoose.connect:

mongoose.Promise = global.Promise;

Ara anem a desenvolupar els controladors per poder gestionar les diferents accions per els usuaris

Crearem la carpeta controllers i elaborarem un arxiu anomenat user.js.

Dins d’aquest arxiu, crearem una funció amb dos parametres req. La primera la emprarem per controlar la peticio i la segona la resposta.

'use strict'

function proves(req, res){

    res.status(200).send({

        message: "Pobant una acció del controlador"

    });

}

module.exports = {

    proves

};

Ara crearem una carpeta per definir les diferents rutes de la nostre app.

'use strict'

var express = require("express");

var UsuariControlller = require("../controllers/usuari");

var api = express.Router();

api.get("/tots", UsuariControlller.proves);

module.exports = api;

Retornem a l’arxiu app.js i Després de la var app = express(); , situarem la carrega de les diferents rutes.

var user\_routes = require("./routes/usuari");

Definim una ruta base sobre la que treballar i la que tindrem que precedir devant de cada peticio. Aquest el tenim que situar devant de la ruta amb el método get que habiem definit en un inici.

app.use("/api", user\_routes);

Ara continuarem elaborant un mètode nou dins d’usuari.js, per guardar un nou usuari i gracias el seguent codi:

1. Carregarem el model implicat en una variable anomenada Usuari.

var Usuari = require("../models/usuari");

1. Després generarem la funcio guardarUsuari()

function guardarUsuari(req, res){

    var usuari = new Usuari();

    var params = req.body; *//recollim totes les dades que arriven en la peticio amb el metode post*

    usuari.nom = params.nom;

    usuari.cognom = params.cognom;

    usuari.email = params.email;

    usuari.clau = params.clau;

    console.log(params);

}

1. Per finalizar afegirem en l’exportació el nou mètode.

module.exports = {

    proves,

    guardarUsuari

};

Ara tenim que afegir la nova ruta en routes/usuari.js, pero definint que és de tipus post i és a on dirigirem les peticions per enregistrar dades.

api.post("/registre", UsuariController.guardarUsuari);

Ara ja podem probar l’enviament de dades amb l’us de postman, realitzant una peticio post a la url amb la ruta definida <http://localhost:3977/api/registre> , amb el format x-www-form-urlencoded i per ultim indiquem les dades d’exemple (nom, cognom, email i clau).

Veurem com rebem aquestes dades en format d’un objecte json, gracias a que es mostraran en la consola de la terminal.

Ara podem completar el mètode guardarUsuari amb la lógica per enmagatzemar les dades rebudes.

*if*(usuari.nom != null && usuari.cognom != null && usuari.email != null && usuari.clau != null ){

        usuari.save((err, usuariStored) => {

*if*(err){

                res.status(500).send({message: "Error al guardar l'usuari"});

            } *else* {

*if*(!usuariStored){

                    res.status(404).send({message: "No s'ha registrat l'usuari"});

                } *else* {

                    res.status(200).send({usuari: usuariStored});

                }

            }

        })

    } *else* {

        res.status(402).send({message: "Indica totes les dades"});

    }

Recordem que hem de definir adicionalment la ruta per aquest controlador, afegint la següent linia en l’arxiu routes/usuari.js

api.post("/registre", UsuariController.guardarUsuari);

Veure l’arxiu controllers/usuari.js

Un altre opció que podem emprar és obtenir les dades d’un usuari mitjantçant el seu email, amb el següent codi:

unction veureUsuari(req, res){

    var params = req.body;

    var email = params.email;

*//var clau = params.clau;*

    Usuari.findOne({email: email.toLowerCase()}, (err, usuari) => { *//Permet cercar un registre per una propietat i tenim que definirli una funció fletxa amb el error i l'objecte*

*if*(err){

            res.status(500).send({message: "Error en la solicitud"});

        }*else*{

*if*(!usuari){

                res.status(404).send({message: "Les credencials són incorrectes"});

            } *else*{

*//Dades del usuari loguejat*

                res.status(200).send({usuari}); *//Si no indiquem cap propietat, emprara la de usuari que coincideix amb el nom de l'objecte.*

            }

        }

    });

}

I adicionalment hem d’afegir la ruta en routes.usuari.js amb la seguent instrucción:

api.post("/veureusuari", UsuariController.veureUsuari);

Veure totes les dades de tots els usuaris amb un altre controlador veureTotsUsuari.

function veureTotsUsuari(req, res){

    var params = req.body;

    var email = params.email;

*//var clau = params.clau;*

    Usuari.find((err, usuari) => { *//Permet cercar un registre per una propietat i tenim que definirli una funció fletxa amb el error i l'objecte*

*if*(err){

            res.status(500).send({message: "Error en la solicitud"});

        }*else*{

*if*(!usuari){

                res.status(404).send({message: "Les credencials són incorrectes"});

            } *else*{

*//Dades del usuari loguejat*

                res.status(200).send({usuaris: usuari}); *//Si no indiquem cap propietat, emprara la de usuari que coincideix amb el nom de l'objecte.*

            }

        }

    });

}

I adicionalment hem d’afegir la ruta en routes.usuari.js amb la seguent instrucción:

api.get("/veuretotsusuari", UsuariController.veureTotsUsuari);

Ara anem a realizar un controlador per poder actualitzar algún parametre d’un usuari i per aixo hem de crear un mètode mes en l’arxiu controllers/usuari.js.

function actualitzarUsuari(req, res){

    var usuariId = req.params.id;

    var update = req.body;

    Usuari.findByIdAndUpdate(usuariId, update, (err,usuariActualitzat) =>{

*if*(err){

            res.status(500).send({message: "Error al actualitzar l'usuari"});

        }*else*{

*if*(!usuariActualitzat){

                res.status(404).send({message: "No s'ha pogut actualitzar l'usuari"});

            }*else*{

                res.status(200).send({usuari: usuariActualitzat});

            }

        }

    });

}

No oblidem el referenciar-ho en l’area de module.exports i a continuació hem de realizar una ruta per aquest controlador.

En aquest cas, inclourem la possibilitat d’afegir el parametre id al final i si situem el simbol ? aconseguirem que sigui opcional, pero en aquest cas no será el nostre objectiu.

api.put("/actualitzar-usuari/:id", UsuariController.actualitzarUsuari);

Ara podem probar en postman la peticio amb el mètode PUT i enviant nomès el parametre que desitgem actualitzar, ja que la resta d’atributs de l’usuari no caldràn.

Ja per ultim ens caldra només el realizar un controlador per borrar un usuari identificantlo amb el seu id i ho podem portar a terme, afegint les seguents linies en l’arxiu controllers/usuari.js

function borrarUsuari(req, res){

    var usuariId = req.params.id;

    var update = req.body;

    Usuari.findByIdAndDelete(usuariId, (err,usuariBorrat) =>{

*if*(err){

            res.status(500).send({message: "Error al actualitzar l'usuari"});

        }*else*{

*if*(!usuariBorrat){

                res.status(404).send({message: "No s'ha pogut borrar l'usuari"});

            }*else*{

                res.status(200).send({usuari: usuariBorrat});

            }

        }

    });

}

No oblidem el referenciar-ho en l’area de module.exports i a continuació hem de realizar una ruta per aquest controlador.

En aquest cas, inclourem la possibilitat d’afegir el parametre id al final i aixi referenciarem quiné s el registre que volem eliminar.

api.delete("/borrarusuari/:id", UsuariController.borrarUsuari);