# 1 iniciar NPM

npm init

## 2 instalar Express

npm install express –save

## 3. crear index.js

Usar editor de texto

## En index js iniciar el archivo con

'use strict' // Desde node 6 se pone esto para el manejo de varaibles

const express= require ('express'); // Llamar a librería express del nomde-modules

## Crear Servidor con Express

const app= express();

app.listen(3000, ()=> {

console.log('API REST corriendo en localhost, port 3000')

});

## Instalar body parser

npm i body-parser –save

## agregar body parser al index .js

const bodyParser= require('body-parser'); //Middleware Capas q se emplean como tubos que conectaran

## Usar body Parser (básico

app.use(bodyParser.urlencoded({extended: false }));

app.use(bodyParser.json());

## instalar Nodemon

permite arrancar el servidor automáticamente al detectar cambios sin necesidad de parar e iniciar el servidor Node.js

npm i –d nodemon -d : Dependencias: devDependencies

## Crear el Script en package.json

En la sección scripts se debe crear el ejecutable de nodemon :

"start": "nodemon index.js"

## Para ejecutar el proyecto usar

Node start index.js

# 2 Insertar Metodo GET

Ejemplo GET /hola

app.get('/hola', (req,res) =>{

res.send({"message": 'Hola Mundo!!!'})

})Ejemplo GET Hola con parámetros

app.get('/hola/:name', (req,res) =>{

res.send({message: `Hola ${req.params.name}!`})

})

Inicia la construcción de Métodos según HTTP

Crear métodos GET y POST para el API de la siguiente forma:

'use strict' //Desde node 6 se pone esto

const express= require ('express');

const bodyParser= require('body-parser'); //Middleware Capas q se emplean como tubos que conectaran

const port= process.env.PORT || 3001;

const app= express();

app.use(bodyParser.urlencoded({extended: false }));

app.use(bodyParser.json());

app.get('/api/product', (req,res) =>{

res.send(200, {products:[]}) //array JSON con productos

});

app.get('/api/product/:productId', (req,res)=> {

});

app.post('/api/product', (req, res)=>{

console.log(req.body);

res.status(200).send({message:'Producto se ha recibido V2'})

});

app.put('/api/product/:productId',(req,res)=>{

}) ;

app.delete('/api/product/:productId',(req,res) =>{

});

app.listen(port, ()=> {

console.log(`API REST corriendo en localhost, puerto: ${port}`)

});

# Configurar MongoDB

## Editar Variables de entorno

Ir a la ruta de Mongo DB

Crear la variable de entorno

En path con la ruta C:\Program Files\MongoDB\Server\3.6\bin

## Crear carpetas Data y db en C:\

## Instalar Driver **mongoose** que tiene un nivel superior

## npm i -s mongoose

const mongoose= require('mongoose');

Crear la conexión a MongoDB con mongoose

mongoose.connect('mongodb://localhost:27017/shop', (err, res)=> {

if(err) {

console.log(`Error al conectar la base de datos: ${err}`)

}

console.log('Connection a la base de datos establecida...')

});

# Crear el Modelo del producto con Moongoose hacer un POST

Crear carpeta models en el directorio raíz

Crear archivo product.js

Inicializar el archivo con:

'use strict'

const mongoose= require('mongoose');

const Schema= mongoose.Schema;

//Creacion del Schema del producto

const ProductSchema= Schema ({

name: String,

picture: String,

price: {type: Number, default: 0},

category: {type: String, enum: ['computers', 'phones', 'accesories','laptop']},

description: String

});

//Exportar modulo de modelo para ser usado por El modulo INDEX

module.exports = mongoose.model('Product', ProductSchema)

Indicar la ruta en el archivo index.js

const Product = require('./models/product') // No es libreria NPM, se pone ruta completa

Modificar el app.post donde se enviaba mediante post los datos del elemento

app.post('/api/product', (req, res)=>{

console.log ('POST/api/product')

console.log (req.body)

//Se crea un objeto de tipo product

let product = new Product()

product.name =req.body.name

product.picture =req.body.picture

product.price =req.body.price

product.category =req.body.category

product.description =req.body.description

product.save((err, productStored)=>{

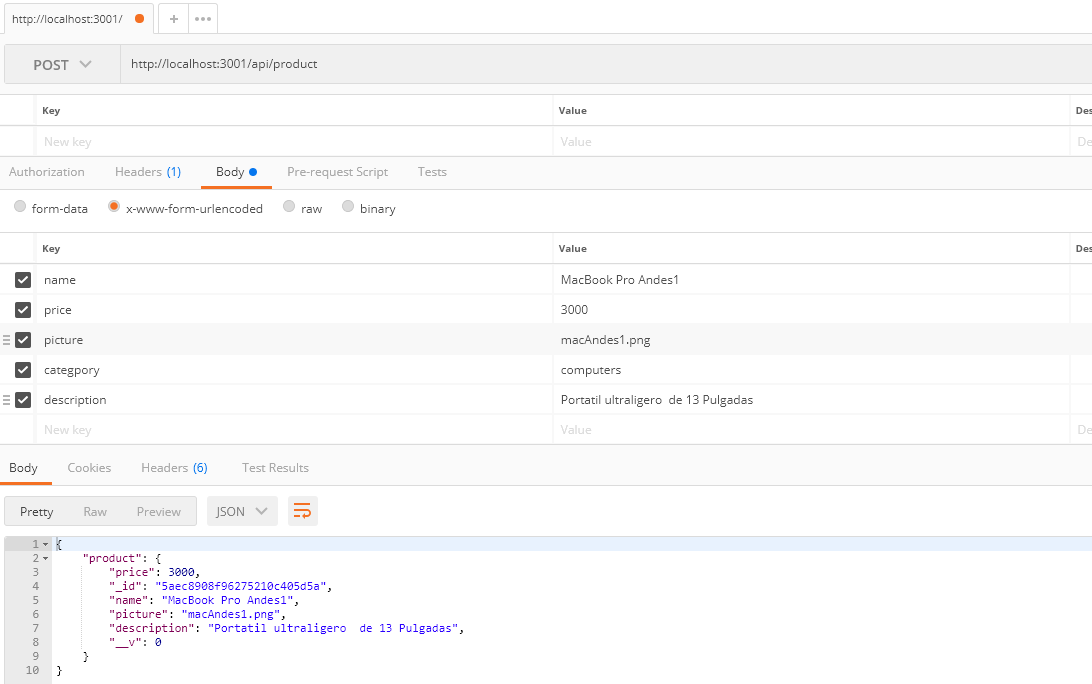
if (err) res.status(500). send({message: `Error al guardar en la base de datos ${err}`})

res.status(200).send({product:productStored})

})

});

Utilizar Postman para el test de los productos A la URL http://localhost:3001/api/product



# Crear el Modelo del producto con Moongoose hacer un GET

-Se modifica la ruta del app.get para introducir la función de Mongoose que maneja los elementos de la DB Mongo

-Se debe realizar el manejo de errores correctamente

-probar con Postman(preferiblemente), retorna un JSON

app.get('/api/product/:productId', (req,res)=> {

let productId= req.params.productId

Product.findById(productId, (err, product)=>{

if(err) return res.status.send({message:`Error al realizar la petición ${err}`})

if(!product) return res.status(404).send({message: `El producto no existe`})

res.status(200).send({product})

})

});

# Crear el Modelo para la consulta de todos los productos

-Se modifica la ruta del app.get para introducir la función de Mongoose que maneja los elementos de la DB Mongo

-Se debe realizar el manejo de errores correctamente

-probar con Postman(preferiblemente), retorna un JSON

app.get('/api/product', (req,res) =>{

Product.find({}, (err, products)=>{

if(err) return res.status.send({message:`Error al realizar la petición ${err}`})

if(!products) return res.status(404).send({message: `No existen productos`})

res.send(200, {products} ) //array JSON con productos

})

});

# Crear el Modelo para borrar un elemento utilizando el método DELETE del HTTP

Se modifica la ruta del app.delete para introducir la función de Mongoose que maneja los elementos de la DB Mongo

-Se debe realizar el manejo de errores correctamente

-probar con Postman(preferiblemente), retorna un JSON

app.delete('/api/product/:productId',(req,res) =>{

let productId= req.params.productId

Product.findById(productId, (err, product)=>{

if(err) res.status(500).send({message: `Error al Borrar el producto ${productId}`})

product.remove(err=>{

if(err) res.status(500).send({message: `Error al Borrar el producto ${productId}`})

res.status(200).send({message: `El producto ha sido eliminado, id: ${productId}` })

})

})

});

# Crear el Modelo para actualizar un elemento utilizando el método PUT del HTTP

app.put('/api/product/:productId',(req,res)=>{

let productId= req.params.productId

let update = req.body

Product.findByIdAndUpdate(productId, update, (err, productUpdated)=>{

if(err) res.status(500).send({message: `Error al actualizar el producto ${productId}`})

res.status(200).send({product: productUpdated})

})

}) ;

Actualmente todo el código se encuentra en un solo fichero (index.js) a partir de ahora se dividirán las funciones del documento para que sea más fácil de realizar el mantenimiento y trabajo a futuro, también la escalabilidad

# Separar el código realizado a la fecha a ficheros divididos por función (refactoriza)

Crear carpeta controllers donde pondrán los controladores requeridos

Crear archivo product.js

Crear funciones con los métodos necesarios para interactuar con la DB según los métodos del HTTP, GET, POST, PUT, DELETE, se debe quitar del index JS y poner en este archivo, se debe exportar los módulos de las funciones creadas.

'use strict'

const Product = require('../models/product') // No es libreria NPM, se pone ruta completa

function getProduct(req, res){

let productId= req.params.productId

Product.findById(productId, (err, product)=>{

if(err) return res.status.send({message:`Error al realizar la petición ${err}`})

if(!product) return res.status(404).send({message: `El producto no existe`})

res.status(200).send({product}) // console.log(product)

})

}

function getProducts(req, res){

Product.find({}, (err, products)=>{

if(err) return res.status.send({message:`Error al realizar la petición ${err}`})

if(!products) return res.status(404).send({message: `No existen productos`})

res.send(200, {products} ) //array JSON con productos

})

}

function saveProduct(req, res){

console.log ('POST/api/product')

console.log (req.body)

let product = new Product()

product.name =req.body.name

product.picture =req.body.picture

product.price =req.body.price

product.category =req.body.category

product.description =req.body.description

product.save((err, productStored)=>{

if (err) res.status(500). send({message: `Error al guardar en la base de datos ${err}`})

res.status(200).send({product:productStored})

})

}

function updateProduct(req, res){

let productId= req.params.productId

let update = req.body

Product.findByIdAndUpdate(productId, update, (err, productUpdated)=>{

if(err) res.status(500).send({message: `Error al actualizar el producto ${productId}`})

res.status(200).send({product: productUpdated})

})

}

function deleteProduct(req, res){

let productId= req.params.productId

Product.findById(productId, (err, product)=>{

if(err) res.status(500).send({message: `Error al Borrar el producto ${productId}`})

product.remove(err=>{

if(err) res.status(500).send({message: `Error al Borrar el producto ${productId}`})

res.status(200).send({message: `El producto ha sido eliminado, id: ${productId}` })

})

})

}

module.exports= {

getProduct,

getProducts,

saveProduct,

updateProduct,

deleteProduct

}

Crear carpeta routes con el archivo index.js donde se pondrán las rutas Con los métodos configurados en el archivo controllers. Se cambiaran las arrow function por funciones creadas en archivo controllers/products.js, se llamara al controlador de productos mediante require para importar sus datos

'use strict'

const express = require ('express')

const productCtrl = require('../controllers/product')

const api = express.Router()

api.get('/product', productCtrl.getProducts);

api.get('/product/:productId', productCtrl.getProduct)

api.post('/product', productCtrl.saveProduct);

api.put('/product/:productId',productCtrl.saveProduct);

api.delete('/product/:productId',productCtrl.deleteProduct);

module.exports= api

Crear archivo app.js donde se pondrán el body parser, Express y el llamado a la API con las rutas (routes.js)

'use strict' //Desde node 6 se pone esto

const express = require ('express');

const bodyParser = require('body-parser'); //Middleware Capas q se emplean como tubos que conectaran

const app = express();

const api = require('./routes')

app.use(bodyParser.urlencoded({extended: false }));

app.use(bodyParser.json());

app.use('/api', api)

// Exportar módulo para utilizar

module.exports = app

Crear archivo config.js donde se configurara el puerto del servidor y la dirección de la DB

'use strict' //Desde node 6 se pone esto

module.exports = {

port : process.env.PORT || 3001,

db : 'mongodb://localhost:27017/shop'

}

El archivo index se modificara de la siguiente manera

'use strict' //Desde node 6 se pone esto

const mongoose = require('mongoose');

const app = require ('./app');

const config = require ('./config');

mongoose.connect(config.db, (err, res)=> {

if(err) { console.log(`Error al conectar la base de datos: ${err}`) }

console.log('Connection a la base de datos establecida...')

});

app.listen(config.port, ()=> {

console.log(`API REST corriendo en localhost, puerto: ${config.port}`)

});

# Crear un modelo con mongoose Schemas.

En la carpeta models crear un archivo users.js

'use strict'

const mongoose = require ('mongoose')

const Schema = mongoose.Schema

//Codoficar Contraseña

const bcrypt = require('bcript-nodejs')

const crypto = require('crypto')

// Crear el Schema utilizando mongoose Tipos de datos, validación y evitar obtener la contraseña cuando se hace una consulta de usuario,

const UserSchema = new Schema ({

email: {type: String, unique:true, lowercase: true},

displayName: String,

avatar: String,

password: {type: String, select: false},

signupDate:{ type:Date, default: Date.now()},

lastLogin:Date

})

// Se crea un método previo al momento de guardar que crea un Password pro defecto,

UserSchema.pre('save', (next)=> {

let user = this

if(!user.isModified('password')) return next ()

bcrypt.genSalt(10,(err, salt )=>{

if (err) return next()

bcrypt.hash(user.password, salt , null, (err, hash)=>{

if (err) return next(err)

user.password=hash

next()

})

})

})

// Se crea un método previo denominado gravatar para establecer un avatar por defecto en un sitio.

UserSchema.methods.gravatar = function () {

if(!this.email) return `https://gravatar.com/avatar/?s=200&d=retro`

const md5=crypto.createHash('md5').update(this.email).digest('hex')

return `https://gravatar.com/avatar/${md5}?s=200&d=retro`

}

// Se exporta el Modelo de usuers

module.exports = mongoose.model('User', UserSchema)

# Autenticación basada en tokens

Para realizar autenticación basada en token se utiliza un elemento muy empleado en la web que se denomina JWT (JSON Web Token), en NPM existe paquetes como jwt-simple

JWT utiliza un encabezado, un cuerpo de mensaje y una firma para el mensaje.

Crear en controllers el archivo auth.js

'use strict'

const mongoose = require ('mongoose')

const User = require('../models/user')

const service = require('../services')

function signUp(req, res){

const user = new User ({

email: req.body.email,

displayName: requ.body.displayName,

})

user.save((err)=>{

if(err) res.status(500).send({message: `Error al crear el usuario: ${err}`})

return res.status(200).send({token: service.createToken(user)})

})

}

function signIn(req,res){

}

module.exports ={

signUp,

signIn

}

Crear una carpeta Services y n archivo index.js con los servicios

'use strict'

const jwt = require('jwt-simpe')

const moment = require('moment')

const config = require('../config')

function createToken(user){

const payload ={

sub: user.id,

iat: moment().unix() ,

exp: moment().add(14, 'days') .unix(),

}

return jwt.encode(payload, config.SECRET\_TOKEN )

}

module.exports= createToken

Modificar config.js

//Agregar SECRET TOKEN

'use strict' //Desde node 6 se pone esto

module.exports = {

port : process.env.PORT || 3001,

db : 'mongodb://localhost:27017/shop',

SECRET\_TOKEN: 'miClaveDeTokens'

}

* Los datos quedan en el package.json