# 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)=>{

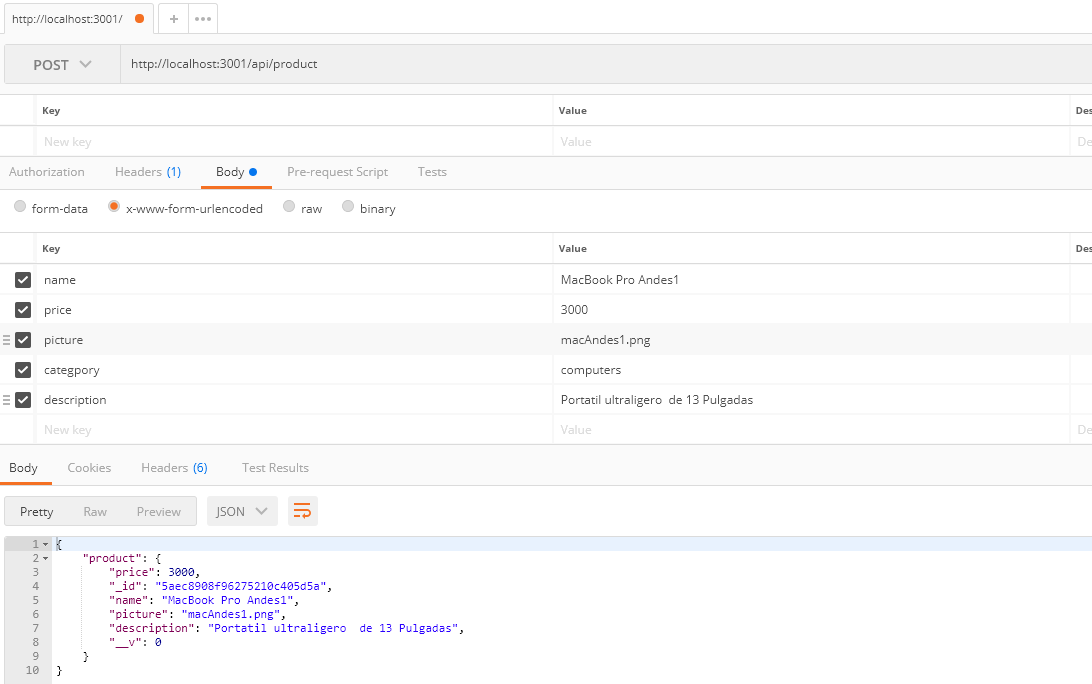
if (err) res.status(500). send({message: `Error al guardar en la base de datos ${err}`})

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}` })

})

})

});

* Los datos quedan en el package.json