**INSTALAR NODE JS**

1. Vamos a la página web e instalamos NodeJs
2. Desde la línea de comando podemos ver la versión con el comando **node-v**
3. Para descargar y consultar la versión de node se utiliza comando **node --version**
4. Para ver la versión del gestor de paquetes tenemos que poner **npm -v**
5. Para ejecutar en node un documento javascript usamos el comando **node -nombreDocumento.js**

**EMPEZANDO A DESARROLLAR UN BACKEND CON NODEJS**

CREAR UN PROYECTO CON NODEJS

1. Utilizamos la línea de comando LINUX (GitBash o Cygwin64)
2. Nos dirigimos a la ubicación C:\Users\MIGUEL\Documents\@CURSOS UDEMY\Máster en JavaScript\NodeJs
3. Creamos una carpeta con el comando **mkdir 02-Proyecto-Nodejs**
4. Lanzamos un proyecto mediante nodeJs con el comando **npm init**, este pedirá información
   1. **"name":** "api-rest-nodejs",
   2. "**version":** "1.0.0",
   3. **"description":** "Api Rest del máster en Javascipt",
   4. **"main":** "index.js",
   5. **"author":** "Miguel Bohorquez",
   6. **"license":** "MIT",

ANADIR DEPENDENCIAS

Creamos varias dependencias que generan mayor facilidad a la hora de generar el proyecto, tenemos que revisar algunas

1. **Express:** Permite definir rutas y dependencias

* Comando: **npm install express --save** (--save significa que salva solo en esta carpeta)

1. **Body-Parser:** Genera las peticiones en archivo JSON

Comando: **npm install body-parser -- save**

1. **connect-multiparty:** Permite subir archivos en el servidor
   * Comando: **npm install connect-multiparty - -save**
2. **Mongoose:** Son métodos para crear modelos y entidades
   * Comando: **npm install mongoose - -save**
3. **Nodemon:** Hace que el código se refresque
   * Comando: **npm install nodemon - -save-dev** (-dev significa que solo estaré presente en desarrollo local)

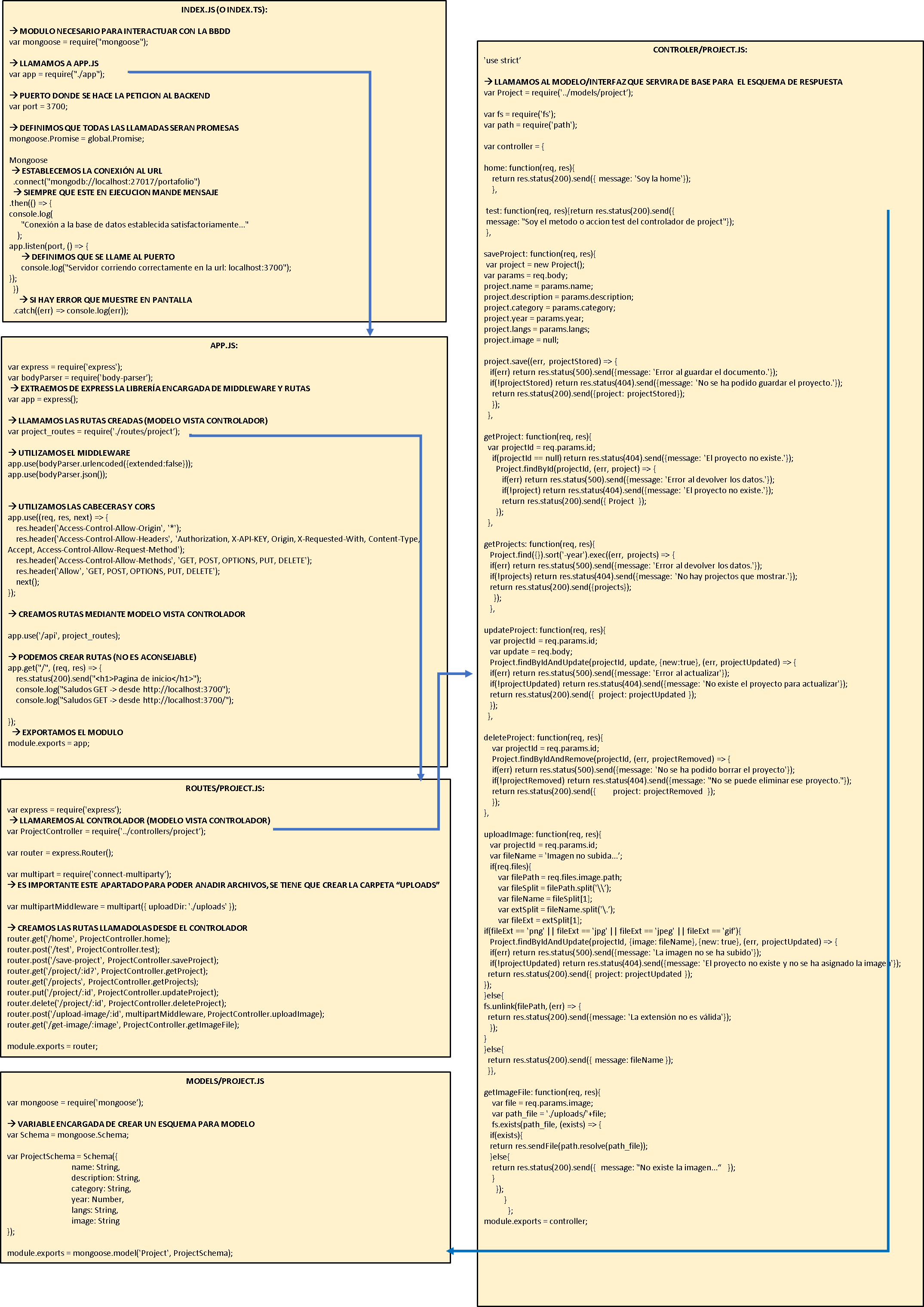
|  |  |
| --- | --- |
| Creamos en la carpeta 02-proyecto-nodejs el archivo ***index.js***; que va a ser el que ejecuta  Dentro de la carpeta package.json la opción de nodemon (para que se actualice automáticamente)  **“start”:”nodemon index.js”** |  |

1. **Os:** Dependencia ya instalada dentro de NPM, podemos ver la información dentro del equipo, no hace falta utilizar ningún comando, más información 🡪 [OS | Node.js v16.3.0 Documentation (nodejs.org)](https://nodejs.org/api/os.html)

|  |
| --- |
| **FUNCTION.JS** |
| const os = require("os"); // Este es el módulo encargado del sistema operativo  var Memoria\_total = os.totalmem();  var Memoria\_Libre = os.freemem();  var Nombre\_host = os.hostname();  var DirectorioLocal = os.homedir();  console.log("La memoria total es: " + Memoria\_total);  console.log("La memoria disponible es: " + Memoria\_Libre);  console.log("Nombre del usuario de windows: " + Nombre\_host);  console.log("Directorio local es: " + DirectorioLocal); |

1. **MODULO FILE SYSTEM (FS):** Es un módulo muy utilizado para tratar documentos, se puede crear varios archivos, no necesita preocuparse por las salidas del directorio. y la api es la misma que el sistema de archivos del nodo. Este no es un costo de tiempo existente para este complemento.Más información 🡪 <https://nodejs.org/api/fs.html>
   * Comando: **npm install file-system –save**

|  |
| --- |
| **FUNCTION.JS** |
| const fs = require("fs");  **🡪 LEEMOS UN DOCUMENTO EN LA TERMINAL**  fs.readFile("data.txt", "utf-8", (error, data) => {      if (error) {          console.log(`Error provocado ${error}`);      }else{          console.log(data);      }  });  **🡪 CAMBIAMOS EL NOMBRE DE UN DOCUMENTO**  fs.rename("data.txt","data\_02.txt",(error=>{  if (error) throw error; // OTRA FORMA DE CONDICIONAL IF, QUE ARROJE ERROR SINO CUMPLE  console.log("Cambio realizado");  }));  **🡪 MODIFICAMOS EL CONTENIDO DEL ARCHIVO:**  fs.appendFile("data.txt","\n Texto cambiado",(error)=>{ // PERMITE CAMBIAR EL DOCUMENTO QUE SE ENCUENTRA DENTRO DEL DOCUMENTO  if (error) throw error; // \n --> ES SALTO DE LINEA  });  **🡪 BORRAMOS UN ARCHIVO:**  fs.unlink("Nombre\_archivo.txt",(error)=>{  if(error) throw error; //SI EXISTE ERROR MUESTRA ERROR  });  **🡪 CREAMOS UNA COPIA DE UN ARCHIVO Y CREAMOS OTRO DE DIFERENTE NOMBRE:**  fs.createReadStream("Nombre\_archivo.txt").pipe(fs.createWriteStream("Archivo\_Copia.txt")); // COPIAMOS UN ARCHIVO A OTRO  **🡪 BUSCAMOS EN UBICACIÓN UN DOCUMENTO, SOLO VEMOS TITULO, LEE DE FORMA ASINCRONA EL CONTENIDO DE UN DIRECTORIO DETERMINADO.**  fs.readdir("/ubicacion/carpeta\_documentos",(error,datos)=>{ // LEEMOS EL ARCHIVO EN TERMINAL, NO LO QUE INCLUYE SINO EL TITULO  datos.forEach( UnDato =>{  console.log(" El titulo del documento es: "+UnDato);  });  }); |



CREAR UNA BASE DE DATOS CON MONGODB DESDE ROBO3T

1. Vamos a crear una nueva base de datos en MongoDb, para ello tenemos que ir a la documentación y descargar el documento.
   * Importante: Tenemos que crear dentro del disco duro C, una ruta en donde irá la base de datos, se crea la carpeta para evitar posibles conflictos **C:/data/db**
   * Mongodb demon siempre se tiene que tener ejecutado en segundo plano mediante la ruta **C:\Program Files\MongoDB\Server\4.4\bin**, también podemos tener ejecutado el mongo.exe que sirve para hacer consultas.
2. Instalamos Robo 3T (o Robomongo), es un gestor de la base de datos, para poderlo utilizar tenemos que instalarlo y tener activo MongoD
   * Nos pedirá nombre, apellido y correo para registrar el usuario
   * Nos pedirá una conexión, utilizaremos la predeterminada

|  |  |  |
| --- | --- | --- |
| **CONNECTION:** | DIRECT CONNECTION |  |
| **LOCALHOST:** | 27017 |
| **CREATE A DATABASE:** | PORTAFOLIO |
| **CREATE COLLECTIONS:** | PROJECTS (SE VA AGREGAR TODA LA INFORMACION DEL PROJECTO) |

CREAR UNA CONEXIÓN A LA BASE DE DATOS CON NODEJS

|  |
| --- |
| **EN INDEX.JS** |
| "use strict"  var mongoose = require('mongoose');  mongoose.set("useFindAndModify", false);  mongoose.Promise = global.Promise;  mongoose.connect("mongodb://localhost:27017/portafolio",  { useNewUrlParser: true, useUnifiedTopology: true })  .then(() => {  console.log("EXITO");  })  .catch((err) => {  console.log(err)  }); |

|  |  |
| --- | --- |
| **var mongoose = require('mongoose');** | Importamos el módulo de mongoose |
| **mongoose.Promise = global.Promise;** | Conexión a la base de datos mediante una Promesa |
| **mongoose.connect("URL",{})** | Creamos una conexión a la base de datos |
| **mongoose.connect("URL",**  **{ useNewUrlParser:true, useUnifiedTopology: true })** | Arregla adventencia de NodeJs |
| **.then ( ()=>{})** | Al ser una promesa NO se usa this sino .then y es callback |
| **.catch( (err)=>{})** | Con el .catch captamos el error la cual es la variable Err |

COMPROBAMOS LA CONEXIÓN DENTRO LA TERMINAL EN CON **$NPM START**

CREAR SERVIDOR:

* Vamos a crear un documento llamado APP.JS:

|  |
| --- |
| **CREAMOS APP.JS** |
| "use strict"  var express = require("express");  var bodyParser = require("body-parser");  var app = express();  // MIDDLEWARE  app.use(bodyParser.urlencoded({ extended: false }));  **🡪 CONFIGURACION NECESARIA PARA BODY PARSER**  app.use(bodyParser.json()); //  **🡪 TODA LA INFORMA QUE LLEGUE LA CONVIERTA EN JSON**  // RUTAS GENERADAS DESDE MODELO-VISTA-CONTROLADOR  app.use("/api", project\_routes);  **// IMPORTANTE SE TIENE QUE UTILIZAR PARA MODELO VISTA CONTROLADOR**  // RUTAS SENCILLAS  app.get("/", (req, res) => {  res.status(200).send("<h1>Pagina de inicio</h1>");  console.log("Saludos GET -> desde http://localhost:3700");  console.log("Saludos GET -> desde http://localhost:3700/");  });  app.get("/test", (req, res) => {  res.status(200).send("<h2>Pagina de Test</h2>");  console.log("Saludos GET -> desde http://localhost:3700/test");  });  app.post("/test", (req, res) => {  console.log("Saludos POST -> desde http://localhost:3700/test");  /\* SE VA A TOMAR TODO LO ANADIDO DEL BODY -> URLENCODED \*/  console.log(**req.body.nombre**);  **🡪 DESDE POST REQ.BODY.NOMBRE\_DEFINIDO\_BODY LLAMA LA INFORMACION**  **QUE SE LANCE EN EL CUERPO DE LA PAGINA**  res.status(200).send("Respuesta correcta en test");  console.log(req.query.web);  **🡪 LANZA LA INFORMACION QUE SE DA DESDE EL URL SEGUIDO DE SIGNO ?**  console.log("Saludos POST -> desde http://localhost:3700/test?web=www.hotmail.com")  });  app.post("/test/:id", (req, res) => {  console.log("Saludos POST -> desde http://localhost:3700/testNumeroID");  console.log("El ID asignado es: " + req.params.id);  **🡪 SE UTILIZA .PARAMS PARA RECOGER UN ID**  res.status(200).send("Respuesta correcta con ID");  });  app.get("/web", (req, res) => {  req.status(200).send("Respuesta correcta con Get -> Web");  console.log("Saludos desde GET -> http://localhost:3700/web");  console.log(req.query.web)  });  module.exports = app;  **🡪 ES NECESARIO EXPORTARLO LO UTILIZAREMOS EN INDEX.JS** |

|  |  |
| --- | --- |
| **var express = require("express");** | Invocamos al módulo de EXPRESS |
| **var bodyParser = require("body-parser");** | Invocamos al módulo de BODY-PARSER |
| **var app = express();** | Creamos una variable con los módulos que están dentro de express |

**MIDDLEWARE**: Es la lógica de intercambio de información entre paquete de datos.

|  |  |
| --- | --- |
| app.use(bodyParser.urlencoded({ extended: false })); | Informa que se acepta cualquier contenido en URL, es necesario para el TESTING. |
| app.use(bodyParser.json()); // | Creamos que la variable APP convierta mediante body-parser todo lo que se extrae a archivos JSON. |

**CREAR RUTAS GET/POST:** Genera los accesos directos

|  |  |
| --- | --- |
| **app.get("/", (req, res) => {**  **res.status(200).send("<h1>TEXTO EN HTML</h1>");**  **console.log("TEXTO EN LINEA DE COMANDO”);**  **});** | **App.get(“URL”,(request,response) =>{}**  Es el formato tradicional de una ruta  **Res.status(200).send();**  Es la información que se envia si estatus OK |

**EXPORTAR MÓDULO:**

|  |  |
| --- | --- |
| **module.exports = app;** | Debemos de exportar el módulo app hacia **INDEX.JS** |

**EN INDEX.JS:**

LLAMAREMOS AL MODULO DE APP.JS, LO INTEGRAREMOS EN INDEX.JS Y CREAREMOS LA CONEXIÓN AL SERVIDOR:

|  |
| --- |
| **INDEX.JS:** |
| 'use strict'  var mongoose = require('mongoose');  var app = require('./app'); **🡪 LLAMAREMOS AL SERVIDOR**  var port = 3700; **🡪 CREAMOS UNA VARIABLE CON EL PUERTO AL QUE SE HARA LA PETICION**  mongoose.Promise = global.Promise;  mongoose.connect('mongodb://localhost:27017/portafolio')          .then(() => {              console.log("Conexión a la base de datos establecida satisfactoriamente...");              // Creacion del servidor              app.listen(port, () => {                  console.log("Servidor corriendo correctamente en la url: localhost:3700");                });          })          .catch(err => console.log(err)); |

|  |  |
| --- | --- |
| **APP.LISTEN( PORT, ()=>{});** | Creamos una conexión a la base de datos mediante el puerto |

**MODELO VISTA CONTROLADOR MCV**

HABLAR DEL MODELO.VISTA.CONTROLADOR

CREAMOS EL MODELO:

Creamos la carpeta MODELS -> Project.ts

Es una plantilla que va a hacer referencia a la base de datos

|  |
| --- |
| **PROJECT.TS (CARPETA MODELS)** |
| "use strict"  var mongoose = require("mongoose");  var Schema = mongoose.Schema;  var ProjectSchema = Schema({  name: String,  description: String,  category: String,  year: Number,  langs: String,  image: String  });  module.exports = mongoose.model("Project", ProjectSchema); |

|  |  |
| --- | --- |
| var mongoose = require("mongoose"); | Llamamos a Mongoose, el cual se encarga de generar esquemas de tipo JSON |
| var Schema = mongoose.Schema; | Generamos un esquema con el nombre de la variable Schema, la cual va a contener el archivo JSON |
| var ProjectSchema = Schema({  name: String,  description: String,  category: String,  year: Number,  langs: String,  image: String  }); | Se genera el JSON que servirá de modelo |
| module.exports = mongoose.model("Project", ProjectSchema); | **.exports** → |
| Genera la opción de exportarlo |
| **.model(“Nombre\_BBDD, NombreVar)** |
| Es el JSON que va a cargar sobre la base de datos generada en MongoBD |

CREAMOS EL CONTROLADOR:

Creamos la carpeta CONTROLER -> Project.ts

El CONTROLADOR es una variable que tiene acciones (métodos) que va a poder hacer el proyecto.

|  |
| --- |
| **PROJECT.TS (CONTROLADOR)** |
| 'use strict'  var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs'); // INVOVAMOS ALA LIBRERÍA FILE SYSTEM PARA CARGAR ARCHIVOS  var path = require('path'); // INVOCAMOS A LA LIBRERÍA PARA REGISTRO DE ARCHIVOS |

1. **INVOCAMOS AL CONTROLADOR:**

Invocamos a una variable Controlador la cual va a generar directivas sobre la URL y las estaremos comprobando con POSTMAN

|  |
| --- |
| var controller = {  home: function (req, res) {  return res.status(200).send({  message: ["Soy Home”, “Puedo dar varios mensajes”]  });  test: function (req, res) {  return res.status(200).send({  message: "Soy Test"  });  },  };  module.exports = controller; |
| Creamos una función para el apartado localhost:3700/api/ home/  Recordamos que para eso nosotros habíamos creado en APP.js |

1. **CREAMOS UN ARCHIVO DE RUTAS (RECOMENDADO)**

|  |
| --- |
| var express = require('express');  // UTILIZAMOS LA LIBRERÍA EXPRESS ENCARGADAS DE HACER API  var ProjectController = require('../controllers/project');  // UTILIZAMOS LOS CONTROLADORES ANTERIORES  var router = express.Router();  // UTILIZAMOS EL MIDDLEWARE PARA INVOCAR AL CONECT-MULTIPARTY PARA SUBIR IMAGENES  var multipart = require('connect-multiparty');  **// UTILIZAMOS LA LIBRERÍA PARA CARGAR ARCHIVOS EN EL SERVIDOR**  var multipartMiddleware = multipart({ uploadDir: './uploads' });  **// VINCULAMOS LA CARPETA DONDE SE ANADEN LOS ARCHIVOS EN EL SERVIDOR**  router.get('/home', ProjectController.home);  router.post('/test', ProjectController.test);  module.exports = router; |
| PARA CONFIGURAR LAS RUTAS PERSONALIZADAS CREAMOS UNA CARPETA ROUTES 🡪 PROJECT.JS |

1. **EN EL SERVIDOR (APP.JS) IMPORTAMOS LAS RUTAS PERSONALIZADAS**

|  |
| --- |
| 'use strict'  var express = require('express');  var bodyParser = require('body-parser'); // LIBRERÍA NECESARIA PARA PETICIONES JSON  var app = express(); // EXTRAEMOS EN UNA VAR LA LIBRERÍA EXPRESS  var project\_routes = require('./routes/project');  // ---- MIDDLEWARE ---  app.use(bodyParser.urlencoded({ extended: false }));  // NECESARIO EN MIDLEWARE PARA TESTING, DEFINE QUE SE UTILIZA CUALQUIER CONTENIDO URL  app.use(bodyParser.json());  // --- CONFIGURAR CABECERAS Y CORS --- (NECESARIO)  app.use((req, res, next) => {  res.header('Access-Control-Allow-Origin', '\*');  res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Allow-Request-Method');  res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');  res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');  next();  });  // --- RUTAS GENERADAS DESDE MODELO-VISTA-CONTROLADOR ---  app.use('/api', project\_routes);  // GENERA QUE SIEMPRE PASEN LAS RUTAS POR **/API** ES LO MAS UTILIZADO Y BUENAS PRACTICAS |
| PARA CONFIGURAR LAS RUTAS PERSONALIZADAS CREAMOS UNA CARPETA ROUTES 🡪 PROJECT.JS |

**METODOS PARA GUARDAR NUEVOS DOCUMENTOS (SAVEPROJECT – project.save() ):**

1. **CREAMOS EL METODO SAVEPROJECT (EN CONTROLLERS – PROJECT.JS)**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  saveProject: function (req, res) {  var project = new Project(); **// GENERA AUTOMATICAMENTE UN ID**  var params = req.body;  project.name = params.name;  project.description = params.description;  project.category = params.category;  project.year = params.year;  project.langs = params.langs;  project.image = null;  project.save((err, projectStored) => {  if (err) return res.status(500).send({ message: 'Error al guardar el documento.' });  if (!projectStored) return res.status(404).send({ message: 'No se ha podido guardar el proyecto.' });  return res.status(200).send({ project: projectStored });  });  },  };  module.exports = controller; | |
| **var project = new Project()**  Generamos una variable con un “new array”  **var params = req.body;**  Tomamos todos los parámetros que contiene el body, en este caso de POSTMAN | |
| **project.name = params.name;**  **project.description = params.description;**  **project.category = params.category;**  **project.year = params.year;**  **project.langs = params.langs;**  **project.image = null;**  Lo que hacemos es reemplazar la información recabada en parámetros de POSTMAN por la que se encontraba en MODELS | |
| project**.save**((err, projectStored) => {};  .save → Permite guardar el archivo  Err → Genera el parámetroConError  projectStored → Genera el parámetroSinError | |
| **if (err) return res.status(500).send({ message: “**Error500**”});** | Se informa del error 500 |
| **if (!projectStored) return res.status(404).send({ message: “**Error400”**});** | Se informa del error 404 |
| **return res.status(200).send({ project:** projectStored **});** | Se informa de éxito status 200 y arroja el parámetro **projectStored** |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.post('/save-project', ProjectController.saveProject);  module.exports = router; |

**METODOS PARA GUARDAR BUSCAR OBJETO POR ID (GETPROJECT – Project.findById):**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  getProject: function (req, res) {  var projectId = req.params.id;  if (projectId == null) return res.status(404).send({ message: "No se ha puesto ID" });  Project.findById(projectId, (err, project) => {  if (err) return res.status(500).send({ message: 'Error al devolver los datos.' });  if (!project) return res.status(404).send({ message: 'El proyecto no existe.' });  return res.status(200).send({ project });  });  }  };  module.exports = controller; | |
| **var projectId = req.params.id;** | Solicitamos un ID de carácter OBLIGATORIO a través de Project.ts → routes |
| **if (projectId == null) {**  **return res.status(404).send(**  **{ message: "No se ha puesto ID" }**  **)**  **};** | **if (projectId == null) {};** |
| Generamos condicional si es valor nulo que arroje un mensaje |
| **return res.status(404).send(**  **{ message: "No se ha puesto ID" }** |
| Generamos que si el error es 404 (siempre lo será) arroje el mensaje |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.get('/project/:id?', ProjectController.getProject);  // :id => ID obligatorio -- :id? => ID opcional  module.exports = router; |

**METODOS PARA BUSCAR POR UN CRITERIO Y ORDENAR (GETPROJECTS – Project.find() );**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  // METODO PARA BUSCAR OBJETO POR UN CRITERIO Y ORDENAR  getProjects: function (req, res) {  Project.find({}).sort('-year').exec((err, projects) => {  // .exec --> ejecuta la funcion de callback  if (err) return res.status(500).send({ message: 'Error al devolver los datos.' });  if (!projects) return res.status(404).send({ message: 'No hay projectos que mostrar.' });  return res.status(200).send({ projects });  });  }  };  module.exports = controller; | |
| **Project.find({“BUSQUEDA”}).sort(“+year”).exec((err,Project)=>{});**  Genera la variable de búsqueda mediante **FIND** | |
| .sort(“búsqueda”) | Genera la variable a buscar dentro del JSON |
| .exec(err,success)=>{} | Ejecuta la función de callback sobre la función |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.get('/projects', ProjectController.getProjects);  module.exports = router; |

**METODOS PARA BUSCAR Y REEMPLAZAR POR CRITERIOS (UPDATEPROJECT – Project.findIdAndUpdate() ):**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  // METODO PARA BUSCAR OBJETO POR UN CRITERIO Y ORDENAR  updateProject: function (req, res) {  var projectId = req.params.id; //PASAR EL ID POR EL URL  var update = req.body; // RECOGE LA PETICION DEL CAMBIO  Project.findByIdAndUpdate(projectId, update, { new: true }, (err, projectUpdated) => {  if (err) return res.status(500).send({ message: 'Error al actualizar' });  if (!projectUpdated) return res.status(404).send({ message: 'No existe el proyecto para actualizar' });  return res.status(200).send({ project: projectUpdated });  });  }  };  module.exports = controller; | |
| **Project.findByIdAndUpdate(ProjectId,update, {new: true}(err,projectUpdate)=>{});** | |
| **{new: true}** | SIN {new: true} Hace que se guarde en la base de datos, pero no lo muestre en pantalla |
| CON {new: true} Hace que se cargue en la base de datos y lo actualice en pantalla |
| **Var projectId = req.params.id;** | Como buscamos por ID desde POSTMAN es necesario localizar el ID a través de una variable |
| **Var update = req.body;** | Dentro de POSTMAN al localizar el URL lo estaremos remplazando, esta información se van a tomar todos los cambios realizados en Body -> x-www-form-urlencoded |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.put('/project/:id', ProjectController.updateProject); //UTILIZAMOS PUT  module.exports = router; |

**METODOS PARA BUSCAR Y BORRAR POR ID (DELETEPROJECT - findIdAndRemove):**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  deleteProject: function (req, res) {  var projectId = req.params.id;  Project.findByIdAndRemove(projectId, (err, projectRemoved) => {  if (err) return res.status(500).send({ message: 'No se ha podido borrar el proyecto' });  if (!projectRemoved) return res.status(404).send({ message: "No se puede eliminar ese proyecto." });  return res.status(200).send({ project: projectRemoved });  });  }  };  module.exports = controller; | |
| **Project.findByIdAndUpdate(ProjectId,update, (err,projectUpdate)=>{});** | |
| **Var projectId = req.params.id;** | Como buscamos por ID desde POSTMAN es necesario localizar el ID a través de una variable |
| **Var update = req.body;** | Dentro de POSTMAN al localizar el URL lo estaremos remplazando, esta información se van a tomar todos los cambios realizados en Body -> x-www-form-urlencoded |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.delete('/project/:id', ProjectController.deleteProject);  module.exports = router; |

**METODO PARA SUBIR IMAGEN ( UploadImage – Project.** **findByIdAndUpdate() ):**

**CREAMOS EL METODO PARA SUBIR IMAGEN (UPLOAD IMAGE):**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  uploadImage: function (req, res) {  var projectId = req.params.id;  var fileName = 'Imagen no subida...'; // ES EL TEXTO LUEGO SE REEMPLAZARA MEDIANTE POSTMAN  if (req.files) {  var filePath = req.files.image.path;  var fileSplit = filePath.split('\\');  var fileName = fileSplit[1];  var extSplit = fileName.split('\.');  var fileExt = extSplit[1];  if (fileExt == 'png' || fileExt == 'jpg' || fileExt == 'jpeg' || fileExt == 'gif') {  Project.findByIdAndUpdate(projectId, { image: fileName }, { new: true }, (err, projectUpdated) => {  if (err) return res.status(500).send({ message: 'La imagen no se ha subido' });  if (!projectUpdated) return res.status(404).send({ message: 'El proyecto no existe y no se ha asignado la imagen' });  return res.status(200).send({ project: projectUpdated });  });  } else {  fs.unlink(filePath, (err) => {  return res.status(200).send({ message: 'La extensión no es válida' });  });  }  } else {  return res.status(200).send({ message: fileName });  }  }  };  module.exports = controller; | |
| **Var Project = require(“Ubicación”);** | Es la ubicación que tenemos para el modelo personalizado (models/projects) |
| **Var fs = require(“fs”);** | Vamos a necesitar File System para poder cargar archivos (REQ.FILE), también necesitamos un middleware |
| **Var path = require(“path”);** | Es una librería necesaria para el registro de archivos |
| **Var controller = {};** | Es el controlador de todos los métodos |
| **uploadImage: function (request,response){};** | Es el método encargado de cargar imágenes |
| **Var projectId = req.params.id;** | Vamos a poner como variable necesaria para cargar la imagen |
| **Var fileName = “Imagen no subida…”;** | Es la variable que va a contener la imagen, le damos un contenido primero |
| **If (req.file){}** | Creamos condicional SI CARGA IMAGEN |
| **Var filePath = req.file.image.path;** | Es la variable donde se genera la ruta  EJEMPLO: upload\\IdPreasignado.jpg |
| **Var fileSplit = filePath.split(“\\”);** | Con .split(“\\”) cortamos parte de la ruta del path, con esto reducimos el nombre preasignado, (SOLO CORTAMOS)  EJEMPLO: IdPreasignado.jpg |
| **Var FileName = fileSplit[1];** | Extraemos de la imagen cargada, solamente la ruta cortada array[1]  EJEMPLO IdPreasignado.jpg |
| **Var extSplit = fileName.split(“\”);** | Volvemos a cortar la ruta del archivo, para sacar la extensión del archivo (SOLO CORTAMOS)  EJEMPLO: .JPG |
| **Var fileExt = extSplit[1];** | Creamos una variable con la parte cortada, es para el condicional IF |
| **If(fileExt ==”jpg” || “png”){}else{}** | Estamos creando la condicional para que solo cargue imágenes |
| **Project.findByIdAndUpdate(projectId, {image:fileName},{new: true}, (err,projectUpdated) =>{});** | Project.findByIdAndUpdate(()=>{})  Es el método encargado de cargar y actualizar la imagen |
| projectId Es el cambio realizado |
| {image: fileName} Es el objeto que vamos a cambiar |
| {new: true} Significa que borre el anterior y actualice el nuevo en pantalla |
| (err, projectUpdated)=>{}  Es el request y response del metodo |
|  |
| **}else{**  **Fs.unlink(filepath,(err)=>{}** | Con la librería de FIlesystem lo que haremos será borrar el documento, sino pasa por el else |
| **} else {**  **return res.status(200).send({ message: fileName });**  **}** | Else creado del primer IF, da información sino se sube archivo con extensión imagen |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.post('/upload-image/:id', multipartMiddleware, ProjectController.uploadImage);  module.exports = router; |

**METODO PARA SOLICITAR IMAGEN DESDE EL BACKEND (getImageFile – fs.exists() ):**

**CREAMOS EL METODO SOLICITAR IMAGEN DESDE EL BACKEND (GETIMAGE):**

|  |  |
| --- | --- |
| **EN CONTROLLERS – PROJECT.JS** | |
| var Project = require('../models/project'); // PARA METODOS PERSONALIZADOS  var fs = require('fs');  var path = require('path');  var controller = {  getImageFile: function (req, res) {  var file = req.params.image;  var path\_file = './uploads/' + file;  fs.exists(path\_file, (exists) => {  if (exists) {  return res.sendFile(path.resolve(path\_file));  } else {  return res.status(200).send({  message: "No existe la imagen..."  });  }  });  }  };  module.exports = controller; | |
| **getImageFile: function (req,res){}** | Es el método que elegimos para poder extraer una imagen del backend |
| **Var file = req.params.image;** | Estamos extrayendo la imagen de los parámetros que tenemos por parte del formulario (pintados en la página) |
| **Var path\_file = “./uploads/”+file;** | Creamos una var con la ubicación del archivo más |
| **Fs.exist(path\_file(err,exist)=>{});** | Utilizamos la librería File System, con el cual comprobamos si existe ruta, se realice callback |
| **If(exist){}** | Comprueba con condicional IF si existe |
| **Return res.sendFile(path.resolve(path\_file));** | var path = require("path");  Es un módulo dentro de NodeJs encargado de poder extraer las rutas dentro del equipo  path.resolve("ubicacion");  Se encarga de devolver el archivo mediante la ubicación seleccionada |

* 1. **CREAMOS LA RUTA DE SAVEPROJECT**

|  |
| --- |
| **EN ROUTES – PROJECT.JS** |
| 'use strict'  var express = require('express');  var ProjectController = require('../controllers/project');  var router = express.Router();  var multipart = require('connect-multiparty');  var multipartMiddleware = multipart({ uploadDir: './uploads' });  router.get('/get-image/:image', ProjectController.getImageFile);  module.exports = router; |

**METODO PARA SUBIR ARCHIVOS [UTILIZANDO CONNECT-MULTIPARTY]**

1) Generamos un modelo que servirá para añadir imagen a la base de datos

2) Generamos la ruta en Routes --> project.ts y utilizamos la variable del Middleware antes del método

router.post("/upload-image/:id",connectMultipartyMiddleware, ProjectController.uploadImage);

3) Para subir imagenes necesitamos invocar al connect-multiparty, lo invocamos desde las las rutas Routes -->project.ts

var connectMultiparty = require("connect-multiparty");

4) Para cargar los documentos necesitamos crear una carpeta [./uploads]

5) Creamos un Middleware para cargar las imagenes

var connectMultipartyMiddleware = connectMultiparty({uploadDir: "./uploads"});

6) Creamos el método para cargar una imagen

uploadImage: function (request, response){}

7) Creamos las variables definidas para localizar las imagenes

var projectId = req.params.id; --> Es el parámetro ID obligatorio para localizar el objeto

var fileName = "Imagen no subida..."; --> Es la variable que contendra la imagen, al no cargar da mensaje predeterminado

8) Generamos el primer condicional si carga o no la imagen if(req.file){}else {}

if (req.files) { --> Generamos condicion SI la request captura un documento "file", con requisito del connector-multiparty

} else { --> Generamos el condicional Sino

return res.status(200).send({ --> Si la respuesta es "correcta" .status(200) envia mensaje

message: fileName --> Envia mensaje variable fileName

});

}

9) Si se cumple la primera condicional se genera lo siguiente

var filePath = req.files.image.path; --> Genera la variable de la ruta donde sube la imagen

var fileSplit = filePath.split("\\"); --> .split("\\") Genera cortar parte del nombre que se preasigna

var fileName = fileSplit[1]; --> Genera el array[1] con la ruta cortada

var extSplit = fileName.split('\.'); --> Volvemos a cortar la ruta con el nombre del archivo

var fileExt = extSplit[1]; --> Volvemos a quedarnos con la ruta cortada

10) Volvemos a generar la condicional para comprobar si lo que se sube es en formato imagen

if (fileExt == "png" || fileExt == "jpg" || fileExt == "gif" || fileExt == "jpeg") {

}else{ }

10.1 ) Si el condicional es correcto se genera los siguientes métodos

Project.findByIdAndUpdate(projectId, { image: fileName }, { new: true }, (err, projectUpdated) => {});

Project.findByIdAndUpdate --> Generamos la opción que encuentre por ID y añada la imagen

projectId --> Es el parámetro ID obligatorio para localizar el objeto

{image: fileName} --> Generamos como parámetro la variable y tipo:image para cargar

{new:true} --> Esta opción genera que reemplace en caso de sobrescribir una nueva imagen

(err,projectUpdated) --> Son las variables por parte del callback, genera el error y el success

10.2) Generamos protocolo HTTP en caso de éxito

if (err) return res.status(500).send({ message: "La imagen no se ha subido" });

if (!projectUpdated) return res.status(404).send({ message: "El proyecto no existe y no se ha asignado imagen" });

return res.status(200).send({ project: projectUpdated });

11) Generamos la condicional en caso de que el documento adjunto no sea en formato imagen

11.1) Llamamos al inicio del archivo a la extension File-system; para poder borrar las imagenes en caso de no cargar correctamente

var fs = require("fs");

11.2) Generamos la función dentro del método

fs.unlink(filePath, (err) => { return res.status(200).send({ message: "extension no valida" }) }

fs.unlink() --> Genera mediante la librería fileSystem en caso de que capture un "unlink " es una promesa se puede continuar con .then()

.then(()=>{}); --> Es opcional por parte de unlink

fs.unlinkSync() --> Genera que se genere el error y espera que finalice todo el proceso, también es promesa

(filePath,(err)=>{}) --> Generamos la variable ruta y solo captamos el error, ya que no se va a generar success

return res.status(200).send({ message: "extension no valida" --> Generamos el mensaje

12)Lo comprobamos, nos vamos a postman con el URL y ponemos en body--> form data --> key: imagen type document y lo anadimos

Pagina 65 -233 crear el frontend