**Alcances del RESTLAB (Comienzo de la Implementación)**

Aquí cubriremos varios temas como:

* Definir los alcances de nuestro RESTServer
* CRUD
* Encriptación de contraseñas
* Validaciones personalizadas
* Creación de roles
* Conexiones con MLAB
* Despliegue de base de datos en la nube
* Conexión con Robo 3T con base de datos en la nube
* Configuración de variables de entorno
* Borrado de archivos
* Eliminado físico de la base de datos
* Eliminación por estado en un campo de la colección

**mongoose (paquete para cone3ctar la bases de datos. Nodejs y mongo)**

*Nos seguramos que este corriendo mongod.exe en windows, en linux, mac ejecutar sudo mongod*

buscar npm mongoose.

*documentación oficial:*[**https://mongoosejs.com/**](https://mongoosejs.com/)

**Intalación: *npm install mongoose --save***

***Configurando la conección***

*En el server.js:*

***const mongoose = require('mongoose');***

***mongoose.connect('mongodb://localhost:27017/cafe', ,{***

***useNewUrlParser:true, //Ent la pagina según para usar las nuevas conexiones***

***useUnifiedTopology:true,***

***},(err, res)=>{***

***if(err) throw err;*** *//Si el error existe, diferente de undefined, lo lanza. Si no existe el error, continua con la ejecución*

***console.log('Base de Datos online');***

***});*** *//El callback sirve para procesar un erro si lo hay o una respuesta si la hay.*

***(ver documentaión)***

*Si el servidor no existe mongoose y mongo la crean, por lo que aunque no exista la bd se puede realizar la conección*

***Estructura***

*Los servicios estarán en server/routes/usuario.js así que creamos en archivo y copiamos todos los servicios (get, puet, etc.) ahí.*

*Los exportamos, pero para eso bastará con exportar el app, ya que el contiene los elementos.*

*El archivo principal (server.js) los importamos pero de forma diferente. Usamos el app que está definido (app = requiere ('express'); app.express) y decimos que queremos usar la ruta:*

*app.use( require ('.routers/usuario.js') ); //importamos y especificamos que lo vamos a usar en el app.*

*Creamos otro ruta con el esquema. server/models/usuario.js En este hacemos la estructura del modelo (tabla) que vamos a insertar, usando:*

***const mongoose = require('mongoose');***

***let Schema = mongoose.Squema;*** *//Esquema para el manejo de la base de datos, es un estandar que Squema sea con S*

***let usuarioSquema= new Squema(***

***{***

***nombre:{*** *//En este caso el nombre se tomará de la BD*

***type: String,*** *//Definimos el tipo, son parametros opcionales*

***required: [true, 'El nombre es necesario']*** *//decimos que es requerido, obligatorio. : true al ponerlos entre corchetes es para poner definir un mensaje personalizado y en su caso no obtener el generico*

***}, //Agregamos todos los que querramos***

***}***

**En routes/usuario importamos el esquema**

***const Usuario = require ('../models/usuario');***

*dentro del post creamos objetos que tendrán la estructura del esquema*

***let usuario = new Usuario({nombre=nombre.body, etc});*** *dentro del constructor mandamos los parametros como un objeto literal, el nombre.body viene son mandados por el post del usuario y usando el modulo body-parser*

*importante ver codigo usuario tanto en model como en router.*

**Validacion de email**

En mongoose: solo necesitamos colocar la condición***unic: true,***

Con esto no se prodrá duplicar el correo pero mandará un error complicado de interpretar por lo que para personalizarlo, usaremos un paquete:

***npm mongoose-unique-validator:*** [***https://www.npmjs.com/package/mongoose-unique-validator***](https://www.npmjs.com/package/mongoose-unique-validator)

*instalamos:* ***npm i mongoose-unique-validator --save***

Lo requerimos: **const uniqueValidator = require ('mongoose-unique-validator');**

Debemos haber agregado la propiedad: **unique: true, al campo que queremos sea unico.**

Indicamos que el esquema debe usar un plugin especifico y mandamos el mensaje personalizado, usamos comillas simples, NO backticks, PATH inserta de donde viene el mensaje.

***usuarioSquema.plugin( uniqueValidator, { menssage: '{PATH} debe de ser unico' });***

**Encriptando la contraseña**

Metodo hash de una sola vía.

Para encriptar usaremos: Bcrypt de la que deberemos instalar la version mas reciente con:

<https://www.npmjs.com/package/bcrypt>

**npm install bcrypt@3.0.2** o posterior. em este caso el 5, basto no especificar la version.

Lo requerimos, ver documentación: **const bcrypt = require('bcrypt');**

**para no regresar el password**

Una opción es hacer:  **usarioDB.password=null;** (usuarioDB es la respuesta la hacer usuario.save)

Pero seguira mandando el campo password como null, para evitar esto trabajaremos con el esquema:

usuarioSquema.methods.toJSON = function(){ //.methods --metodos de schema , .toJSON siempre se llama antes de hacer una impresión JSON de schema

//No usar función de flecha porque no permite el uso de this, el cual necesitaremos

let user = this;

let useObject = user.toObject();

delete userObject.password;

return userObject; Con esto eliminamos el campo del esquema,

}

**ACTUALIZANDO UN REGISTRO CON PUT**

Para esto usaremos mongosee, hay varias opciones, usaremos **findByIdAndUpdate.**

**app.put('/usuario/:id', function (req, res) {**

**let id = req.params.id;**

**let body = req.body;**

**//usuario.findById(id, (err,usuarioDB)=>{ //.findById viene del mongoose y es para buscar el usuario aunque despues deberiamos hacer algo como usuarioDB.save**

**usuario.findByIdAndUpdate( id, body, { new: true }, (err,usuarioDB)=>{ //Este busca por el ID y actualiza si lo encuentra. ver documentación**

**//{new} Son opciones de la function y especifica que quiero que devuelva el archivo modificado, de otra forma devuelve el original**

**if(err){**

**return res.status(400).json(**

**{**

**ok:false,**

**err**

**});**

**}**

**res.json({**

**ok: true,**

**usuario: usuarioDB**

**});**

**}); //Fin del findByIdAndUpdate**

**});// fin del put**

**Validaciónes en el PUT (de los ROLES)**

Aunque en el modelos hayamos definido los roles, en la actualización acepta cualquier cosa que se le ponga, para evitar esto debemos pasarle ciertas opciones a mongoose.

usuario.findByIdAndUpdate( id, body, { new: true, **runValidators=true** }, (err,usuarioDB)=>{ //Este busca por el ID y actualiza si lo encuentra. ver documentación, opciones

También debemos evitar actualizar la bandera de google (Solo cuando se registra el susario por google) y tampoco queremos que se actulice el password de esa manera, por lo que haremos.

***Forma 1*** (No recimendado por ser ineficiente, al menos que sea solo uno o dos campos):

**let body. req.body; //Obtenemos el body de la respuesta**

**delete body.password;** //lo eliminamos antes de llamar findByIdAndUpdate para actualizar

**delete body.google;**

***Forma 2***

Usaremos uderscore para filtrar.

los requerimos: const \_ = require('undercore');

usaremos \_.pick() devuelve un bjeto con los paremetros que especificamos en una lista blanca.

**let body = \_.pick(req.body,['nombre','email','img','role','estado']); //**EL body viene de req.body que es donde se encuentran los parametros obtenidos del put

/\*Regresa una copia del objeto con los parametros deseados pasados a traves de una lista blanca\*/

**Implementación del get (Consultas, retorno de los datos de la BD)**

Dentro de la base de datos llamaremos usando el esquema (Usuario) usaremos le metodo find, (existen varios: findByID, find..., etc) solo find regresará todos los usuarios de la BD.

app.get('/usuario', function (req, res) {

let desde = req.query.desde || 0; //Tomamos el valor de desde que se manda como parametro adicional en la url con ?desde= si existe o cero

  desde = Number(desde);

  let limite = req.query.limite || 5;

  limite= Number(limite);

  Usuario.find({}, 'nombre email role estado google img') //devuelve todo el registro, se puede filtrar los que estan activos, por google=true, role:'ADMIN\_ROLE' etc. El lugando parametro opcional indica que campos quiero que retorne

    .skip(desde)  //Indica desde que resgistro empezar y los demás los salta

    .limit(limite)  //Sirve para el número de registros deseados, podriamos mandar una constante pero estoy recibiendolo como un parametro aadicional. con url?limite=5&otro=algo

    .exec((err,usuarios)=>{

        if (err)

        {

          return res.status(400).json({

            ok:false,

            err

          });

        }

        Usuario.count({}, (err,conteo)=>{ //Aquí  porque cuando haga la petición, si es correcta se hace el conteo y se retorna toda la infomación. Aquí también se puede mandar una opción del conteo e {}

          res.json({

            ok:true,

            usuarios,//Significa usuarios: usuarios, es decir la propiedad usuario muestra los usuarios recibidos

            registros\_totales: conteo

          });

        });

      })  //Es como la instrucción ejecutalo

  }); //Fin del get

*Para mandar parámetros adicionales usamos* url?limite=5&otro=algo en el front

*Agregando la cantidad de registros: usamos esquema.count({filtro}, callback);*

**Borrando un elemento (Implementación delete)**

Existen dos maneras, una es borrando físicamente los registros (que ya no se suele hacer a menos que sea necesario, esto para mantener referencias correctas de historiales, etc. ) y otra es cambiando el estado.

*Para esto puede ser una petición post y tomar el valor del body pero en este caso lo recogeremos de url mandándolo como parámetro adicional de la petición get.*

**(recomendable) cambiar el estado de un elemento.**

*Podemos utilizar entre otras opciones el usuario.findByIdAndUpdate con el filtro adecuado. Bastará con modificar en:*

let estado= {'estado':'true'};

  Usuario.find(estado, 'nombre email role estado google img')

Y en …

Usuario.count(estado, (err,conteo)=>{

***Base de datos en la nube***

*Una opción en mongoAtlas (consultar documentación). La opción que usaremos es mLab en su opción gratuita:* [*https://mlab.com*](https://mlab.com) *(Mlab ahora en* **mongo atlas***, así que es lo que usaremos).*

*Configurando la BD*

*Elegimos la opción (gratuita) -> el servicio, trabajaremos con aws -> La región más cercana (free tier avaible)*

*Configuraciones en seguridad – security – network acces*

En add IP acces podemos configurar los ip permitidos a nuestra base de datos, aquí agregaremos el ip de nuestro servidor (el de heroku en este caso) add ip address y podemos agregar las ip’s, por lo pronto permitiremos una conexión desde cualquier ip (acces from anywhere).

Data bases use-> Creamos un nuevo usuario, estos serán permitidos ingresar a la base de datos.

*Seleccionamos permisos (atlas admin) creamos un nombre y contra (generamos).*

*Usuario: somnus*

*Contra: BHxJHn2g2Iyy6FdU*

*MongoDB (URL) Ruta: mongodb+srv://somnus:<password>@cluster0.cbsyw.mongodb.net/cafe*

***Conectando a mongoDB atlas***

*Para esto usaremos mongoDB compass que es el software recomendado.*

*Clusters -> connect -> connect with MongoDB Compass*

Copiamos la cadena de conexión proporcionada y la usamos en compass en *new coneccion para conectarnos a la BD*

*URL a la base de datos: MongoDB (URL) Ruta: mongodb+srv://somnus:<password>@cluster0.cbsyw.mongodb.net/café en este caso queda:*

*mongodb+srv://somnus:BHxJHn2g2Iyy6FdU@cluster0.cbsyw.mongodb.net/cafe*

*Trae la opción de usar la url tal cual, solo agregando el password en su lugar y usando la base de datos correspondiente o podemos llenar de forma individual los campos para lo cual usaremos*

***cluster0.cbsyw.mongodb.net*** *como hostname*

***verificar que la versión de node (mongosee especificamente) este actualizada para no tener problemas de coneccion***

*ir al package.json para ver las versiones*

***npm update*** actualiza todas las dependencias

*En el curso había un problema con actualizaciones de bcrypt por lo que se desinstalará y reinstalará, aunque no tuve dicho problema, lo realizaremos para practicar.*

*npm uninstall bcrypt*

*npm install bcrypt --save*

***Conectando la base de datos en mongo atlas con nuestra app***

**En el archivo config.js**

//ENTORNO

process.env.NODE\_ENV = (!process.env.NODE\_ENV)? 'dev': process.env.NODE\_ENV;  //Variable que establece heroku y si existe algo es que corre en producción, de lo contrario es desarollo, pudo hacers como el PORT

//Base de datos

if(process.env.NODE\_ENV === 'dev') //Para usarlo en la conexión ya sea local o remota

{

    console.log('Estor en dev');

    process.env.URLDB = 'mongodb://localhost:27017/cafe';

}else{

    console.log('Estor en producc');

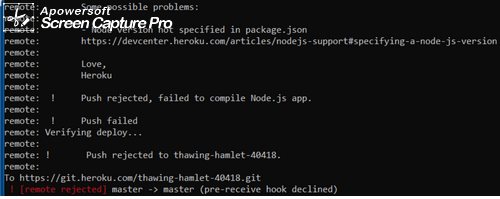
    process.env.URLDB = 'mongodb+srv://somnus:BHxJHn2g2Iyy6FdU@cluster0.cbsyw.mongodb.net/cafe';

}

**Subida a heroku**

**Nota de actualización:**

En el siguiente video, aprenderemos como desplegar nuestra aplicación de Node en Heroku Server... a muchos les funciona sin problemas así como está el video, pero parece que algunos les muestra el siguiente error al intentar desplegar el código:



Si leen un posible error, es porque no especificamos la versión de Node que queremos que corra Heroku. Para especificar la versión de node, podemos hacer esto:

Detectar la versión de node que usamos en el desarrollo



Luego, colocar esa versión en el package.json

***comandos***

*Git add .*

*git commit -m "RestServer - usuario: con mongoDB, moongoose, mongo atlas"*

*heroku login*

*heroku git:remote -a vast-spire-87155 (Solamente si se perdió la configuración anterior, esta es la ruta)*

*git push heroku master**// Para subir a heroku, ya antes se había hecho la configuración*

*heroku open para que nos mande directo a la pagina*

***conectando el robo3t para consultar la base de datos remota (mongo atlas)***

clusters conection, elegimos mongodb compass, copiamos el url

***mongodb+srv://somnus:<password>@cluster0.cbsyw.mongodb.net/test***

***En robo 3t***

En type: replica set, pegamos el enlace donde dice from srv y presionamos el botón, copiará de forma automáticas los cluster. Eliminamos el enlace que copiamos, posteriormente configuramos authentication.

authentication

Database: café (Nombre de la BD)

User Name: somnus Password: *BHxJHn2g2Iyy6FdU*

**Subiendo a git**

En caso de se haya perdido la configuración habría que volver a agregar la ruta

* git remote add origin https://github.com/rusel-arceo/node-restserver.git (Se toma de la página de git, en este caso, repositorio existente)
* git remote (opcional): muestra los repositorios remotos.
* Subimos: git push -u origin master

Como todo ya está configurado solo hacemos ***git push***

**Haciendo el tag**

Averiguamos el último tag

Git tag

Git tag -a v0.0.3 -m “Usuario y restserve base de datos mongoDB y mongoose”

Git push –tags

**Variables de entorno en Heroku (para que solo se vean con introduciendo las credenciales)**

En la terminal:

**Heroku config** Nos muestra las variables de entorno

Heroku config:set nombre=”Rusel” //Esto crea una variable de entorno

Heroku config:get nombre //Obtiene la variable de entorno

Heroku config:unset nombre //Elimina la variable de entorno

Protegiendo el url remoto

Heroku config:set MONGO\_URI = ”mongodb+srv://somnus:BHxJHn2g2Iyy6FdU@cluster0.cbsyw.mongodb.net/cafe”

//Volvemos a url remoto una variable de entorno en heroku

En el código (En el archivo config.js) reemplazamos el url por la variable de entorno process.env.MONGO\_URI

Lo subimos a heroku

Git commit -am “Agregando la variable de entorno MONGO” cuando solo hay un cambio, comprobar con git status (que solo es un cambio, de los contrario usar add, y luego commit)

Git push heroku master

***Subimos a github***

Git push origin master

*NOTAS:*

*Para captura codigo desde visual code Polacode, o carbon.*

WARNIG: Hay un warning “**(node:6536) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.**” Para solucionarlo:

mongoose.connect('mongodb://localhost:27017/cafe', {

  useNewUrlParser: true,  //Ent la pagina según para usar las nuevas conexiones

  useUnifiedTopology:true,

  useCreateIndex: true 🡨-Agregamos

}